Introduction to Programming

Collection of solutions to programming exercises as part of Introduction to Programming (EN), taught by Magnus Madsen.

Contents

Classes

Week 1

Exercise 1.1.5

```
public class UseArgument {
  public static void main(String[] args) {
    System.out.print("Hi, ");
    System.out.print(args[0]);
    System.out.println(". How are you?");
  }
}
```

Class 1: UseArgument

Describe what happens if you try to execute `UseArgument` with each of the following command lines:

a. `java UseArgument java`

   Prints `java` as the name: `Hi, java. How are you?`

b. `java UseArgument @!&^%`

   This will return different results depending on the shell you're using. If you pass those arguments directly as-is, it will just print `Hi, @!&^%. How are you?`.

   If your shell uses any of these characters, not all arguments may be passed to the java program, you may get an error, or get no output. For example, in a POSIX-compatible shell like bash, `&` is used as a delimiter to run programs asynchronously (return user control before the program finishes running), and otherwise acts as `;`. An error may be thrown due to `^%` not being a known command.

c. `java UseArgument 1234`

   Prints `Hi, 1234. How are you?`

d. `java UseArgument.java Bob`

   Newer Java versions run `javac` if a `.java` file is provided as the class name to run, and then run the compiled bytecode.

e. `java UseArgument Alice Bob`

   Arguments are whitespace delimited (in most shells), and only the first one is read in the program: `Hi, Alice. How are you?`

Exercise 1.1.6

Modify `UseArgument.java` to make a program `UseThree.java` that takes three names as command-line arguments and prints a proper sentence with the names in the rev

```
public class UseThree {
  public static void main(String[] args) {
    if (args.length != 3) {
      System.out.println("Error: Program expects exactly three arguments");
      return;
    }
    System.out.printf("Hi %s, %s and %s. How are you?%n", args[2], args[1], args[0]);
  }
}
```

Class 2: UseThree

Week 2

Exercise 1.2.25

Wind chill. Given the temperature T (in degrees Fahrenheit) and the wind speed v (in miles per hour), the National Weather Service defines the effective temperature (the wind chill) as follows:

$w = 35.74 + 0.6215T + (0.4275T - 35.75)v^{0.16}$

Write a program that takes two double command-line arguments temperature and velocity and prints the wind chill. Use `Math.pow(a, b)` to compute a^b. Note: The formula is not valid if T is larger than 50 in absolute value or if v is larger than 120 or less than 3 (you may assume that the values you get are in that range).

```
class WindChill {
  public static void main(String[] args) {
    if (args.length < 2) {
      System.err.println("Usage: [temperature] [velocity]");
      return;
    }
    var temperature = Double.parseDouble(args[0]);
    var velocity = Double.parseDouble(args[1]);
    if (Math.abs(temperature) > 50. || velocity > 120. || velocity < 3.) {
      System.err.println("Values outside of allowed range.");
      return;
    }
    var wind_chill = WindChill.calculateWindChill(temperature, velocity);
    System.out.printf("The effective temperature is %fF.%n", wind_chill);
  }

  public static double calculateWindChill(double T, double v) {
    return 35.74 + (0.6215 * T) + (0.4275 * T - 35.75) * Math.pow(v, 0.16);
  }
}
```

Class 3: WindChill

Exercise 1.2.30

Uniform random numbers. Write a program that prints five uniform random numbers between 0 and 1, their average value, and their minimum and maximum values. Use `Math.random()`, `Math.min()`, and `Math.max()`.

```
import java.util.ArrayList;

class RandomValues {
  public static void main(String[] args) {
    var numbers = new ArrayList<Double>(5);
    for (var i = 0; i < 5; i++)
      numbers.add(Math.random());
    var min = 1.;
    var max = 0.;
    var sum = 0.;
    for (var num : numbers) {
      min = Math.min(num, min);
      max = Math.max(num, max);
      sum += num;
    }

    var avg = sum / numbers.size();

    for (var num : numbers) {
      System.out.printf("%f ", num);
    }
    System.out.println();
    System.out.printf("Average: %f, Minimum: %f, Maximum: %f%n", avg, min, max);
  }
}
```

Class 4: RandomValues

Week 3

Exercise 1.3.5

Write a program `RollLoadedDie` that prints the result of rolling a loaded die such that the probability of getting a 1, 2, 3, 4, or 5 is $\frac{1}{8}$ and the probability of getting a 6 is $\frac{3}{8}$.

```
public class RollLoadedDie {
  public static void main(String[] args) {
    double rand8 = Math.random() * 8.;
    // floor returns double. Round gives long for double input, so conversion needed
    int random = (int)(Math.floor(rand8) + 1);
    // 1 - 8 to 1 - 6, with 6 receiving the probabilities of 7 - 8
    // 1 - 5 have their original probabilities, meaning each have 1/8 chance of being
selected, while 6 is 3/8
    random = Math.min(random, 6);
    System.out.println(random);
  }
}
```

Class 5: RollLoadedDie

Exercise 1.3.24

Write a program `GamblerPlot` that traces a gambler's ruin simulation by printing a line after each bet in which one asterisk corresponds to each dollar held by the gambler.

```
public class GamblerPlot {
  public static void main(String[] args) {
    // Run trials experiments that start with
    // $stake and terminate on $0 or $goal.
    int stake = Integer.parseInt(args[0]);
    int goal = Integer.parseInt(args[1]);
    double probability = 0.5;
    int bets = 0;

    // Run one experiment.
    int cash = stake;
    while (cash > 0 && cash < goal) {
      // Simulate one bet.
      bets++;
      if (Math.random() < probability)
        cash++;
      else
        cash--;

      printCash(cash);
    }

    boolean won = cash == goal;
    String wonText = won ? "Won" : "Lost";

    System.out.println();
    System.out.printf("%s after %s bets", wonText, bets);
  }

  public static void printCash(int cash) {
    for(int i = 0; i < cash; i++) {
      System.out.print("*");
    }
    System.out.println();
  }
}
```

Class 6: GamblerPlot

Exercise 1.3.25

Modify `Gambler` to take an extra command-line argument that specifies the (fixed) probability that the gambler wins each bet. Use your program to try to learn how this probability affects the chance of winning and the expected number of bets. Try a value of p close to 0.5 (say, 0.48).

Small changes of probability affect win % a lot, for example the chance of doubling money from 100 to 200 is roughly 50% for $p = 0.5$, but falls to less than 5% for $p = 0.49$ and 0% below that.

```
public class Gambler {
  public static void main(String[] args) {
    // Run trials experiments that start with
    // $stake and terminate on $0 or $goal.
    int stake = Integer.parseInt(args[0]);
    int goal = Integer.parseInt(args[1]);
    int trials = Integer.parseInt(args[2]);
    double probability = Double.parseDouble(args[3]);
    int bets = 0;
    int wins = 0;
    for (int t = 0; t < trials; t++) {
      // Run one experiment.
      int cash = stake;
      while (cash > 0 && cash < goal) {
        // Simulate one bet.
        bets++;
        if (Math.random() < probability)
          cash++;
        else
          cash--;
      }
      // Cash is either 0 (ruin) or $goal (win).
      if (cash == goal)
        wins++;
    }
    System.out.println(100 * wins / trials + "% wins");
    System.out.println("Avg # bets: " + bets / trials);
  }
}
```

Class 7: Gambler

Week 4

Exercise 1.4.10

Write a program `Deal` that takes an integer command-line argument `n` and prints `n` poker hands (five cards each) from a shuffled deck, separated by blank lines.

A second class `Card` (Class 9) is used for visuals.

Example run for 2 hands (`java Deal.java 2`):

```
J    J K    K 𝟏𝟎♥ ♥𝟏𝟎 9♦ ♦9 7♣ ♣7
♥    ♥ ♣    ♣   ♥♥♥    ♦♦♦    ♣♣♣
♥    ♥ ♣    ♣   ♥♥♥    ♦ ♦    ♣ ♣
J    J K    K 𝟏𝟎♥ ♥𝟏𝟎 9♦ ♦9 7♣ ♣7

3 ♥ 3 𝟒♠ ♠4 𝟒♦ ♦4 7♦ ♦7 6♦ ♦6
  ♥                 ♦♦♦   ♦ ♦

3 ♥ 3 𝟒♠ ♠4 𝟒♦ ♦4 7♦ ♦7 6♦ ♦6
```

```
import java.util.Random;

public class Deal {
  public static void main(String[] args) {
    int handCount = Integer.parseInt(args[0]);
    // We have 52 cards, each hand is 5 cards
    if(handCount > 10) {
      System.err.println("Too many hands! Maximum is 10.\njava Deal [hands]");
    }

    // Generate the deck
    Card[] deck = new Card[52];
    for(var i = 0; i < 52; i++) {
      // 13 cards for each suit, take advantage of int division
      var value = (i % 13) + 1;
      var suit = i / 13;
      deck[i] = new Card(Card.Suit.values()[suit], value);
    }

    // Randomly shuffle the deck
    Random random = new Random();
    for(var i = 0; i < deck.length; i++) {
      // swap each item with a random item in the array
      // isn't truly random, but we're not using crypto random anyway
      var second = random.nextInt(deck.length);
      var a = deck[i];
      deck[i] = deck[second];
      deck[second] = a;
    }

    for(var handNumber = 0; handNumber < handCount; handNumber++) {
      var offset = handNumber * 5;
      // since the deck is shuffled, we just take the next 5 cards from the array
      // java could use an array slice so that we didn't need to copy all the
objects..
      Card[] hand = {
        deck[offset],
        deck[offset+1],
        deck[offset+2],
        deck[offset+3],
        deck[offset+4]
      };
      System.out.println(Card.sprintCards(hand));
    }
  }
}
```

Class 8: Deal

```
public class Card {

  public enum Suit {
    Clubs,
    Diamonds,
    Hearts,
    Spades;

    public static final String CLUBS = "♣";
    public static final String DIAMONDS = "♦";
    public static final String HEARTS = "♥";
    public static final String SPADES = "♠";

    public String sprint() {
      switch(this) {
        case Clubs: return CLUBS;
        case Diamonds: return DIAMONDS;
        case Hearts: return HEARTS;
        case Spades: return SPADES;
      }
      throw new NullPointerException();
    }
  }

  public Suit suit;
  public int value;

  public Card(Suit suit, int value) {
    this.suit = suit;
    this.value = value;
  }

  // Print a single character for the value
  // Only a single character is returned so that we can format
  // the card output correctly and easily
  public char sprintValue() {
    assert this.value > 0 && this.value < 14;
    if(this.value == 1) {
      return 'A';
    } else if(this.value == 11) {
      return 'J';
    } else if(this.value == 12) {
      return 'Q';
    } else if(this.value == 13) {
      return 'K';
    } else if(this.value == 10) {
      // UTF-8 character that looks like a 10
      // but uses only one character width
      return '𝟏𝟎';
    }
    // asserted 0 < x < 14, we handled 1, 10-13
    // the only valid values here are 2-9
    return Integer.toString(this.value).charAt(0);
  }

  // Print the card to a string to be later processed (or printed).
  // Approximates 'normal' card deck look
  // Numbered cards have their suit symbol repeated based on their value.
  // Face cards are empty.
  public String sprintCard() {
    var output = "";
    var value = this.sprintValue();
    var suit = this.suit.sprint();

    // Generate the top (and bottom) of a card
    // this will show the value of the card on each edge
    var top = "";
    top += value;
    if(this.value > 10) {
      // face cards are empty
      top += "   ";
    } else {
      // and for numbered cards, show the suit characters
      top += this.value >= 4 ? suit : " ";
      top += this.value < 4 && this.value > 1 ? suit : " ";
      top += this.value >= 4 ? suit : " ";
    }
    top += value;
    top += "\n";

    output += top;

    if(this.value > 10) {
      // face cards get suits on the side, right above and under their values
      // numbered cards are empty on their sides
      output += suit + "   " + suit + "\n";
      output += suit + "   " + suit + "\n";
    } else {
      // normal cards have either 3 or 4 rows of suits in 1-3 columns,
      // with the middle one sometimes floating.
      // we have to have a set size and can't have floating characters,
      // so this is a best effort approximation
      // instead of using 3 rows we have a gap in the 3rd row
      output += " ";
      output += this.value >= 6 ? suit : " ";
      // odd or 10 have a symbol in the middle
      output += this.value % 2 == 1 || this.value == 10 ? suit : " ";
      output += this.value >= 6 ? suit : " ";
      output += " ";
      output += "\n";


      output += " ";
      output += this.value >= 8 ? suit : " ";
      output += this.value == 10 ? suit : " ";
      output += this.value >= 8 ? suit : " ";
      output += " ";
      output += "\n";
    }

    output += top;

    return output;
  }

  // Shows cards next to each other (left to right)
  // for visuals, assumes that sprintCard returns the same width for each card (and
each row of a card)
  // for correctness, assumes that sprintCard always returns 4 rows (is asserted)
  public static String sprintCards(Card[] cards) {
    String[] output = { "", "", "", "" };

    // split each card into it's 4 rows
    // save the row into relevant output
    for(var i = 0; i < cards.length; i++) {
      var cardstr = cards[i].sprintCard().split("\n");
      assert output.length == cardstr.length;
      for(var x = 0; x < cardstr.length; x++) {
        output[x] += cardstr[x] + " ";
      }
    }

    // and join the rows together with a newline
    var outputstr = "";
    for(var i = 0; i < output.length; i++) {
      outputstr += output[i] + "\n";
    }
    return outputstr;
  }
}
```

Class 9: Card

Exercise 1.4.14

Write a code fragment to print the transposition (rows and columns exchanged) of a square two-dimensional array.

Example input	Output
99 85 98 98 57 78 92 77 76	99 98 92 85 57 77 98 78 76

Table 1: Example input and output matrix

```
public class Transpose {
  public static void main(String[] args) {
    int[][] input = {
      {99, 85, 98},
      {98, 57, 78},
      {92, 77, 76},
    };

    for(var i = 0; i < input.length; i++) {
      for(var j = i; j < input.length; j++) {
        var og = input[i][j];
        input[i][j] = input[j][i];
        input[j][i] = og;
      }
    }

    for(var i = 0; i < input.length; i++) {
      for(var j = 0; j < input.length; j++) {
        System.out.print(input[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```

Class 10: Transpose

Week 5

Exercise 1.5.7

Write a program that takes an integer command-line argument n, reads in $n - 1$ distinct integers between 1 and n, and determines the missing value.

```java
package week5;

import common.StdIn;

public class MissingInt {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    boolean[] nums = new boolean[n];
    for(var i = 0; i < n - 1; i++) {
      int integer = StdIn.readInt() - 1;
      nums[integer] = true;
    }
    for(var i = 0; i < nums.length; i++) {
      if(!nums[i]) {
        System.err.println(i + 1);
      }
    }
  }
}
```

Class 11: MissingInt

Exercise 1.5.19

Write a program that takes as command-line arguments an integer n and a floating-point number p (between 0 and 1), plots n equally spaced points on the circumference of a circle, and then, with probability p for each pair of points, draws a gray line connecting them.

```java
package week5;

import common.StdDraw;

public class CircleLines {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    double p = Double.parseDouble(args[1]);
    assert p >= 0. && p <= 1.;
    StdDraw.setXscale(-1.1, 1.1);
    StdDraw.setYscale(-1.1, 1.1);
    StdDraw.clear();
    double[][] points = new double[n][2];

    for(var i = 0; i < n; i++) {
      // explicitly convert to double
      double fi = i;
      // divide by total to get 0-1
      fi /= n;
      // and convert to radians
      fi *= 2*Math.PI;
      // save calculations
      points[i][0] = Math.sin(fi);
      points[i][1] = Math.cos(fi);
      StdDraw.filledCircle(points[i][0], points[i][1], 0.01);
    }

    StdDraw.setPenColor(StdDraw.GRAY);

    for(var i = 0; i < n; i++) {
      for(var x = i; x < n; x++) {
        if(Math.random() > p) continue;

        StdDraw.line(points[i][0], points[i][1], points[x][0], points[x][1]);
      }
    }
  }
}
```

Class 12: CircleLines

Exercise 1.5.26

Write a program Circles that draws filled circles of random radii at random positions in the unit square, producing images like those below. Your program should take four command-line arguments: the number of circles, the probability that each circle is black, the minimum radius, and the maximum radius.

```java
package week5;

import common.StdDraw;

public class Circles {
  public static void main(String[] args) {
    var circleCount = Integer.parseInt(args[0]);
    var pBlack = Double.parseDouble(args[1]);
    var minRadius = Double.parseDouble(args[2]);
    var maxRadius = Double.parseDouble(args[3]);

    StdDraw.setScale();
    StdDraw.clear();

    for(var i = 0; i < circleCount; i++) {
      if(Math.random() < pBlack) {
        StdDraw.setPenColor(StdDraw.BLACK);
      } else {
        StdDraw.setPenColor(StdDraw.WHITE);
      }
      var radii = Math.random() * (maxRadius - minRadius) + minRadius;
      var posX = Math.random();
      var posY = Math.random();
      StdDraw.filledCircle(posX, posY, radii);
    }
  }
}
```

Class 13: Circles

Week 6

Exercise 2.1.19

Write a static method `histogram()` that takes an `int array a[]` and an integer `m` as arguments and returns an array of length `m` whose `i`th element is the number of times the integer `i` appeared in `a[]`. Assuming the values in `a[]` are all between 0 and `m - 1`, the sum of the values in the returned array should equal `a.length`.

```
package week6;

public class Histogram {
  public static int[] histogram(int[] a, int max) {
    int[] output = new int[max];
    for(var x = 0; x < a.length; x++) {
      output[a[x]]++;
    }
    return output;
  }
}
```

Class 14: Histogram

Exercise 2.1.30

Calendar. Write a program `Calendar` that takes two integer commandline arguments `m` and `y` and prints the monthly calendar for month `m` of year `y`, as in this example:

```
% java Calendar 2 2009
February 2009
 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

```
package week6;

public class Calendar {
  static final String[] months = {
      "January",
      "February",
      "March",
      "April",
      "May",
      "June",
      "July",
      "August",
      "September",
      "October",
      "November",
      "December"
  };

  public static void main(String[] args) {
    int month = Integer.parseInt(args[0]);
    int year = Integer.parseInt(args[1]);

    System.out.printf("%s %d\n", months[month - 1], year);
    System.out.println(" S  M Tu  W Th  F  S");
    int daysToSkip = dayOfWeek(year, month, 1);
    int monthLength = monthLength(year, month);
    for(var i = 1; i < monthLength + daysToSkip + 1; i++) {
      if(i - daysToSkip && i % 7 == 0) {
        System.out.println();
        continue;
      }
      if(i <= daysToSkip) {
        System.out.print("   ");
        continue;
      }
      System.out.printf("%2d ", i - daysToSkip);
      if(i % 7 == 0) {
        System.out.println();
      }
    }
  }

  static final int[] monthLengths = {
      31,
      28,
      31,
      30,
      31,
      30,
      31,
      31,
      30,
      31,
      30,
      31,
  };

  /**
   * Length of a given month
   * @param year
   * @param month 1=January
   * @return length of the month in days, leap year adjusted
   */
  static int monthLength(int year, int month) {
    return monthLengths[month - 1] + (month == 2 ? (isLeapYear(year) ? 1 : 0) : 0);
  }

  static boolean isLeapYear(int year) {
    boolean isLeapYear;
    isLeapYear = (year % 4 == 0);
    isLeapYear = isLeapYear && (year % 100 != 0);
    isLeapYear = isLeapYear || (year % 400 == 0);
    return isLeapYear;
  }

  /**
   * @param year
   * @param month 1=January
   * @param day
   * @return day of week, 0=Sunday,6=Saturday
   */
  static int dayOfWeek(int year, int month, int day) {
    int y0 = year - (14 - month) / 12;
    int leapAdjustedYear = y0 + y0 / 4 - y0 / 100 + y0 / 400;
    int m0 = month + 12 * (14 - month) / 12 - 2;
    return (day + leapAdjustedYear + (31 * m0) / 12) % 7;
  }
}
```

Class 15: Calendar

Exercise 2.2.26

Poker analysis. Write a `StdRandom` and `StdStats` client (with appropriate static methods of its own) to estimate the probabilities of getting one pair, two pair, three of a kind, a full house, and a flush in a five-card poker hand via simulation.

Divide your program into appropriate static methods and defend your design decisions. *Extra credit :* Add straight and straight flush to the list of possibilities.

```
package week6;

import common.StdDraw;
import common.StdRandom;
import common.StdStats;

public class PokerAnalysis {
  /**
   * Example program using PokerAnalysis
   *
   * Renders a plot of each type of hand
   * The bars represent types in Hand.Type, in the same order
   */
  public static void main(String[] args) {
    var decks = 1000;
    // var hands = decks * 10;
    var types = analyzeShuffledDecks(decks);
    var typeFloats = new double[types.length];
    var max = StdStats.max(types);
    for(var i = 0; i < types.length; i++) {
      // max or hands can be used here
      typeFloats[i] = (double)types[i] / max;
    }

    StdDraw.setPenColor();
    StdStats.plotBars(typeFloats);
  }

  /**
   * Gets nth (5-card) hand of a given deck.
   * Up to 10 hands can be dealt from a given deck
   */
  public static Hand getHand(Card[] deck, int handOffset) {
    var offset = handOffset * 5;
    return new Hand(new Card[]{
        deck[offset],
        deck[offset+1],
        deck[offset+2],
        deck[offset+3],
        deck[offset+4]
    });
  }

  /**
   * Gets the first hand of a given deck.
   */
  public static Hand getHand(Card[] deck) {
    return getHand(deck, 0);
  }

  public static Card[] getRandomDeck() {
    Card[] deck = new Card[52];
    for(var i = 0; i < 52; i++) {
      var value = (i % 13) + 1;
      var suit = i / 13;
      deck[i] = new Card(Card.Suit.values()[suit], value);
    }
    StdRandom.shuffle(deck);
    return deck;
  }

  /**
   * Shuffles n decks, draws 10 hands from each, and saves the number of hands of each type
   * Returns an array mapping type (using their ordinals) to number of hands found
   */
  public static int[] analyzeShuffledDecks(int decks) {
    var types = new int[Hand.Type.values().length];
    for(var i = 0; i < decks; i++) {
      var deck = getRandomDeck();
      for(var offset = 0; offset < 10; offset++) {
        var hand = getHand(deck, offset);
        types[hand.type().ordinal()]++;
      }
    }
    return types;
  }

  public static int numberOfHandsOfType(int[] analyzedShuffledDecks, Hand.Type type)
  {
    return analyzedShuffledDecks[type.ordinal()];
  }
}
```

Class 16: PokerAnalysis

```
package week6;

import java.util.Arrays;

public class Hand {
  enum Type {
    StraightFlush,
    FourOfAKind,
    FullHouse,
    Flush,
    Straight,
    ThreeOfAKind,
    TwoPair,
    Pair,
    HighCard;
  }

  Card[] cards;

  public Hand(Card[] cards) {
    assert cards != null;
    assert cards.length == 5;
    this.cards = cards;
  }

  public Type type() {
    var doubleHistogram = doubleHistogram();
    var hasFlush = hasFlush();
    var hasStraight = hasStraight();
    if(hasFlush && hasStraight) return Type.StraightFlush;
    if(hasFlush) return Type.Flush;
    if(hasStraight) return Type.Straight;
    if(doubleHistogram[4] == 1) return Type.FourOfAKind;
    if(doubleHistogram[3] == 1 && doubleHistogram[2] == 1) return Type.FullHouse;
    if(doubleHistogram[3] == 1) return Type.ThreeOfAKind;
    if(doubleHistogram[2] == 2) return Type.TwoPair;
    if(doubleHistogram[2] == 1) return Type.Pair;
    return Type.HighCard;
  }

  int[] values() {
    var values = new int[cards.length];
    for(var i = 0; i < cards.length; i++) {
      values[i] = cards[i].value;
    }
    return values;
  }

  /**
   * Generates a histogram of card values.
   * Each element in an array contains the number of cards with that value.
   */
  int[] histogram() {
    return Histogram.histogram(values(), 14);
  }
  /**
   * Generates a histogram of the histogram of card values.
   * This shows how many times did any repetitions repeat.
   * For example {1,1,2,2,2,3,3} first histogram results in {0,2,3,2}
   * Second histogram (what this function returns) results in {1,0,2,1}.
   * This tells us that there are two pairs and one three of a kind.
   */
  int[] doubleHistogram() {
    // if cards are dealt properly, the max is 5 (cards.length),
    // as a card can't appear more than 4 times
    var histogram = histogram();
    // System.out.println(Arrays.toString(histogram));
    return Histogram.histogram(histogram, cards.length + 1);
  }

  boolean hasFlush() {
    var suit = cards[0].suit;
    for(var i = 1; i < cards.length; i++) {
      if(cards[i].suit != suit) return false;
    }
    return true;
  }

  boolean hasStraight() {
    var values = values();
    Arrays.sort(values);
    var isStraight = true;
    for(var i = 1; i < values.length; i++) {
      if(values[i] != values[i-1] + 1) {
        isStraight = false;
        break;
      }
    }
    if(isStraight) return true;
    return isStraight || Arrays.equals(values, new int[]{
      // A 10 J Q K is valid as well
      1, 10, 11, 12, 13
    });
  }
}
```

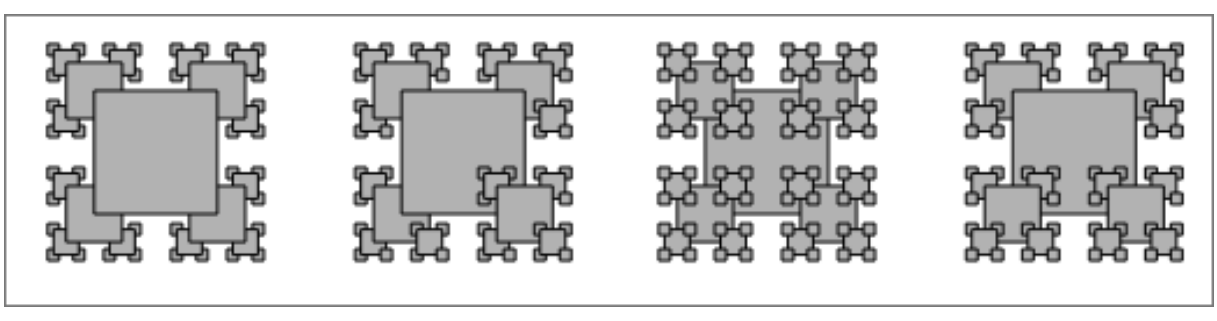
Class 17: Hand

Note that this reuses Class 9 from week 4.

Week 7

Exercise 2.3.22

Recursive squares. Write a program to produce each of the following recursive patterns. The ratio of the sizes of the squares is 2.2 : 1. To draw a shaded square, draw a filled gray square, then an unfilled black square.



```
package week7;

import common.StdDraw;

class Squares {
    public static void main(String[] args) {
        StdDraw.enableDoubleBuffering();
        StdDraw.setCanvasSize(512 * 4, 512);
        StdDraw.setXscale(0, 4);
        drawSquares(0.5, 0.5, .25, 4, 1);
        drawSquares(1.5, 0.5, .25, 4, 2);
        drawSquares(2.5, 0.5, .25, 4, 3);
        drawSquares(3.5, 0.5, .25, 4, 4);
        StdDraw.show();
    }

    public static void drawSquares(
            double x,
            double y,
            double halfLength,
            int limit,
            int mode) {
        var shouldRecurse = limit > 0;
        limit--;
        var newHalfLength = halfLength / 2.2;

        if (mode == 3 || (mode > 1 && !shouldRecurse))
            drawSingleSquare(x, y, halfLength);

        if (shouldRecurse) {
            int[][] offsets = { { 1, 1 }, { -1, 1 }, { -1, -1 }, { 1, -1 } };
            for (var i = 0; i < 4; i++) {
                if (mode == 2 && i == 3)
                    drawSingleSquare(x, y, halfLength);
                if (mode == 4 && i == 2)
                    drawSingleSquare(x, y, halfLength);
                drawSquares(
                        x + halfLength * offsets[i][0],
                        y + halfLength * offsets[i][1],
                        newHalfLength,
                        limit,
                        mode);
            }
        }

        if (mode == 1)
            drawSingleSquare(x, y, halfLength);
    }

    private static void drawSingleSquare(double x, double y, double halfLength) {
        StdDraw.setPenColor(StdDraw.GRAY);
        StdDraw.filledSquare(x, y, halfLength);
        StdDraw.setPenColor();
        StdDraw.square(x, y, halfLength);
    }
}
```

Class 18: Squares

Exercise 2.3.27

Sierpinski triangles. Write a recursive program to draw Sierpinski triangles (see PROGRAM 2.2.3). As with Htree, use a command-line argument to control the depth of the recursion.

```
package week7;

import common.StdDraw;

public class Triangles {
    public static void main(String[] args) {
        int order = Integer.parseInt(args.length > 0 ? args[0] : "1");
        StdDraw.enableDoubleBuffering();

        drawRootTriangle();

        if (order > 1)
            // vertical alignment to center it properly within the root triangle
            recurse(order - 1, 0.5, (height(1)) / 2 - 0.15, 0.5);

        StdDraw.show();
    }

    public static void recurse(int limit, double x, double y, double width) {
        limit--;
        var height = height(width);
        drawTriangle(x, y, width, height);
        var halfWidth = width / 2;
        if (limit > 0) {
            recurse(limit, x, y + height * .75, halfWidth);
            recurse(limit, x - halfWidth, y - height / 4, halfWidth);
            recurse(limit, x + halfWidth, y - height / 4, halfWidth);
        }
    }

    /**
     * Calculates equilateral triangle's height from it's width / side length
     */
    public static double height(double width) {
        return 0.5 * Math.sqrt(3) * width;
    }

    public static void drawTriangle(double x, double y, double width, double height)
{
        var topLeft = new double[] { x - width / 2, y + height / 2 };
        var topRight = new double[] { x + width / 2, y + height / 2 };
        var bottom = new double[] { x, y - height / 2 };
        StdDraw.polygon(new double[] { topLeft[0], topRight[0], bottom[0] },
                new double[] { topLeft[1], topRight[1], bottom[1] });
    }

    /**
     * Variant of draw triangle, but draws it upside down (base/water-level side is
     * down)
     */
    public static void drawRootTriangle() {
        var width = 1;
        var height = height(width);
        var offset = (1 - height) / 2;
        StdDraw.polygon(
                new double[] { 0, 1, 0.5 },
                new double[] { offset, offset, height + offset });
    }
}
```

Class 19: Triangles

Exercise 2.3.31

Plasma clouds. Write a recursive program to draw plasma clouds, using the method suggested in the text.

```
package week7;

import common.StdDraw;
import common.StdRandom;

public class PlasmaClouds {
    public static void main(String[] args) {
        var hurst = Double.parseDouble(args[0]);
        var s = Math.pow(2, 2 * hurst);
        var variance = 0.01;

        StdDraw.enableDoubleBuffering();

        draw(.5, .5, 1, StdRandom.uniformDouble(), StdRandom.uniformDouble(),
StdRandom.uniformDouble(), StdRandom.uniformDouble(), variance, s);

        StdDraw.show();
    }

    // c = color ("color vertical horizontal")
    // t = top
    // r = right
    // l = left
    // m = middle (horizontal)
    // c = center (vertical)
    public static void draw(double x, double y, double halfSize, double ctl, double
ctr, double cbl, double cbr, double var, double s) {
        var color = (ctl+ctr+cbl+cbr) / 4;
        if (halfSize < 1/512.) {
            var clamped = Math.clamp(color, 0, 1);
            StdDraw.setPenColor((int) (clamped * 255), (int) (clamped * 255), 255);
            StdDraw.filledRectangle(x, y, halfSize, halfSize);
            return;
        }
        var newColor = StdRandom.gaussian(color, Math.sqrt(var));
        var newVar = var / s;
        var newHalfSize = halfSize / 2;

        var ctm = (ctl + ctr) / 2;
        var cbm = (cbl + cbr) / 2;
        var ccl = (cbl + ctl) / 2;
        var ccr = (cbr + ctr) / 2;

        // tl
        draw(x - newHalfSize, y - newHalfSize, newHalfSize, ctl, ctm, ccl, newColor,
newVar, s);
        // tr
        draw(x + newHalfSize, y - newHalfSize, newHalfSize, ctm, ctr, newColor, ccr,
newVar, s);
        // bl
        draw(x - newHalfSize, y + newHalfSize, newHalfSize, ccl, newColor, cbl, cbm,
newVar, s);
        // br
        draw(x + newHalfSize, y + newHalfSize, newHalfSize, newColor, ccr, cbm, cbr,
newVar, s);
    }
}
```

Class 20: PlasmaClouds

Week 9

Exercise 3.1.26

Find a website that publishes the current temperature in your area, and write a screen-scraper program Weather so that typing java Weather followed by your ZIP code will give you a weather forecast.

- You can use: https://www.flotvejr.dk/%C3%A5rhus/observations
- You may use city names instead of zip code
- Note: Do not overcommit; we are expecting something simple.

```java
import java.io.FileDescriptor;
import java.io.FileOutputStream;
import java.io.PrintStream;

import common.In;

public class Weather {
    public static void main(String[] args) {
        try {
            // this wasn't needed on my main NixOS system but was needed on SteamOS
            // without this it just printed ? for utf-8 characters
            System.setOut(new PrintStream(new FileOutputStream(FileDescriptor.out),
true, "UTF-8"));
        } catch (Exception e) {
        }

        if (args.length != 1) {
            System.out.println("Missing argument for city name");
            return;
        }
        String url = String.format("https://www.flotvejr.dk/%s/observations",
args[0]);
        // String url = "./test.html";
        String source = new In(url).readAll();
        UglierSoup soup = new UglierSoup(source);

System.out.println("┌────────────┬─────────────────────────────┬────────────┐");
        var isFirst = true;
        for (var row : soup.getElements(".nearby-observations-table tr")) {
            if (isFirst)
                isFirst = false;
            else

System.out.println("├────────────┼─────────────────────────────┼────────────┤");
            var temperature = row.getElement(".nearby-observations-
temperature").getInnerText().trim();
            var location = row.getElement(".nobr a").getInnerText();
            var observation =
row.getElement(".observation_ago").getInnerText().trim().split(" ");
            var time = observation[1] + " mins ago";
            var place = observation[4] + " km away";
            var nobrs = row.getElements(".nobr");
            var windspeed = nobrs[nobrs.length - 1].getInnerText().trim();
            System.out.printf(
                    "│ %-10s │ %-27s │ %-10s │\n│            │ %-12s - %-12s │            
│\n",
                    temperature, location, windspeed, time, place);
        }

System.out.println("└────────────┴─────────────────────────────┴────────────┘");
    }
}
```

Class 21: Weather

Note: Other used classes omited for clarity in PDF form, see ZIP or git source. Includes a simple HTML parser and tag-name/classes selector queries.

Example output:

5.8°	Oedum 91 mins ago - 16 km away	1 m/s
9.1°	Sletterhage Fyr 91 mins ago - 20 km away	7 m/s
4.7°	Tirstrup 31 mins ago - 30 km away	2 m/s
4.7°	Horsens/Bygholm 91 mins ago - 42 km away	3 m/s
5.0°	Hald V 91 mins ago - 46 km away	2 m/s
9.8°	Roesnaes 91 mins ago - 61 km away	9 m/s
4.9°	Isenvad 91 mins ago - 64 km away	2 m/s
9.2°	Gniben 91 mins ago - 69 km away	9 m/s
5.0°	Karup 31 mins ago - 69 km away	2 m/s
7.1°	Odense / Beldringe 31 mins ago - 76 km away	3 m/s
4.8°	Aars Syd 91 mins ago - 79 km away	0 m/s
5.6°	Billund Lufthavn 41 mins ago - 81 km away	1 m/s
8.6°	Aarslev 91 mins ago - 95 km away	3 m/s
6.4°	Vamdrup 31 mins ago - 98 km away	3 m/s
6.5°	Mejrup 91 mins ago - 99 km away	2 m/s
8.1°	Borris 91 mins ago - 100 km away	2 m/s
7.8°	Askov 91 mins ago - 103 km away	2 m/s
9.8°	Assens/Toroe 91 mins ago - 103 km away	4 m/s
8.7°	Holbaek 91 mins ago - 106 km away	5 m/s
5.4°	Aalborg 31 mins ago - 107 km away	3 m/s

Exercise 9.1

Write a class `Person` to represent a person. The class should have the following fields: `String firstName`, `String lastName`, `int age`, and `Person spouse`. The `spouse` field is initially `null`.

Add two constructors:

- `Person(String first, String last)`
- `Person(String first, String last, int age)`

Add getters and setters:

- Add getters for all fields.
- Add setters, but only for lastName and spouse.

Add a method `void birthday()` which increases the persons age by one year.

Add a method `boolean marry(Person that)` which marries this person to that person by updating the `spouse` fields and joining their last names. For example: If Nathan Cole and Emily Parker are married they become Nathan Cole-Parker and Emily Cole-Parker.

- A person cannot be married until they are 18 years old.
- A person cannot be married if they are already married.

The method should return `true` if the marriage is successful.

Add a `toString` method that returns a `String` of the form `Lucky Luke, 23, unmarried`.

Add a `main` method to test your implementation.

```java
public class Person {
    final String firstName;
    String lastName;
    int age;
    Person spouse;

    public Person(String first, String last) {
        this(first, last, 0);
    }
    public Person(String first, String last, int age) {
        this.firstName = first;
        this.lastName = last;
        this.age = age;
    }

    public String getFirstName() {
        return firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public int getAge() {
        return age;
    }
    public Person getSpouse() {
        return spouse;
    }
    public void setSpouse(Person spouse) {
        this.spouse = spouse;
    }

    public void birthday() {
        this.age += 1;
    }

    public boolean marry(Person that) {
        if(this.spouse != null || that.spouse != null) return false;
        if(this.age < 18 || that.age < 18) return false;
        if(this == that) return false;
        this.spouse = that;
        that.spouse = this;

        var lastName = String.format("%s-%s", this.lastName, that.lastName);
        this.lastName = lastName;
        that.lastName = lastName;
        return true;
    }

    public String toString()  {
        var isMarried = this.spouse == null ? "unmarried" : "married";
        return String.format("%s %s, %s, %s", this.firstName, this.lastName,
this.age, isMarried);
    }

    public static void main(String[] args) {
        // check example toString
        assert (new Person("Lucky", "Luke", 23)).toString().equals("Lucky Luke, 23,
unmarried");

        var nathan = new Person("Nathan", "Cole", 17);
        var emily = new Person("Emily", "Parker", 18);

        // not 18
        assert !nathan.marry(emily);
        assert !emily.marry(nathan);

        // is 18
        nathan.birthday();

        assert nathan.marry(emily);
        // marry must fail if married
        assert !nathan.marry(emily);
        assert !emily.marry(nathan);
        assert nathan.lastName.equals("Cole-Parker");
        assert emily.spouse == nathan;
        assert nathan.spouse == emily;
    }
}
```

Class 22: Person

Exercise 9.6

A student was asked to solve the following exercise:

> Write a class `ShoppingCart` to represent a shopping cart in an online store. The class should store the cart owner's name, an array of item names, an array of item prices, and track how many items are in the cart. Include a constructor, methods to add items, remove items, calculate the total price, and apply a discount.

The student wrote the following code:

```java
class cart {
    public String n; // name, maybe unused?
    protected double total; //total price of items
    public String[] items; // this is the items

    public void add(String item, double p) {
        items[count] = item; prices[count] = p; count++;
        total = total + p;
        return;
    }

    // count how many items are in the shopping cart
    int size;

    cart(String n, int size) {
        n = n;
        this.size = this.size;
        //initialise the fields of the class.
        items = new String[100];
        prices = new double[100];
        items = new String[100];
    }

    // Getter method
    public double getTotal() { return total; }

    // count how many items are in the shopping bag
    public static int count = 0; // counter variable
    private double[] prices; //stores the cost

    public void remove(String x) {
        for(int i=0;i<count;i++){
    if(items[i].equals(x)){
        total=total-prices[i];
                for(int j=i;j<count-1;j++){
            items[j]=items[j+1];
            prices[j]=prices[j+1];
        }
                 count--;
        return;
            }}
    }

    public String discount(   double persent) {
        if (persent > 0) {
            total = (double) ((double) total - ((double) total * (double) persent));
            } else {
                total = total - (total * persent);
        } return null; }}
```

- What do you think of the code style?

  Meaningless indentation and (lack of) use of whitespace makes the code hard to follow. Indescriptive naming of variables (`n`, `total`) may be confusing to users of this class (and later to anyone reading it), especially since the comments aren't written in such a way to show on hover on relevant properties (Javadoc). Ordering of methods and properties hides their relationship and makes it harder to skim through the class. Total being calculated when items are added or when discount is applied makes operations order sensitive (applying discount before adding items results in no effect). Not using a subclass/object for linking names/prices of items makes cart manipulation prone to errors, but that's part of the exercise, similar to using arrays instead of vectors/arraylist.

  Persent, if read as percent, may imply the function takes a percentage, but it expects a 'direct' value (90% = 0.90). Some fields being public may allow (accidental) logic errors from class users (e.g. desynchronized items and prices).

- Do you find the comments helpful?

  Better naming of variables would result in less comments needed. Some are self describing and thus only add to the visual noise.

- Refactor the code such that it follows best practices.

```java
class ShoppingCart {
    public String ownerName;
    // assume maximum of 100 items in a cart.
    private String[] itemNames = new String[100];
    private double[] itemPrices = new double[100];
    private int itemCount = 0;
    public double discount = 0.;

    ShoppingCart(String ownerName) {
        this.ownerName = ownerName;
    }

    public double getTotal() {
        var price = 0.;
        for(int i = 0; i < this.itemCount; i++) {
            price += this.itemPrices[i];
        }
        return price * (1. - discount);
    }

    public void add(String name, double price) {
        assert this.itemCount <= this.itemNames.length;
        this.itemNames[this.itemCount] = name;
        this.itemPrices[this.itemCount] = price;
        this.itemCount++;
    }

    /**
     * Removes an item with a given name and price combination from the shopping
cart.
     * Returns boolean of whether such item existed before (and was removed).
     */
    public boolean remove(String name, double price) {
        for (int i = 0; i < this.itemCount; i++) {
            if (this.itemNames[i].equals(name)) {
                if(this.itemPrices[i] != price) continue;
                // overwrite the removed item by the next element
                // results in array with no gaps
                for (int j = i; j < this.itemCount - 1; j++) {
                    this.itemNames[j] = this.itemNames[j + 1];
                    this.itemPrices[j] = this.itemPrices[j + 1];
                }
                // deinitialize last value to prevent data corruption
                // in case of logic errors with e.g. not using itemCount later
                this.itemNames[this.itemCount - 1] = null;
                this.itemPrices[this.itemCount - 1] = 0.;
                this.itemCount -= 1;
                return true;
            }
        }
        return false;
    }

    /**
     * Applies a new discount, overwriting any previously set discount.
     * @param newDiscount as real value, for example '0.1' meaning a 10% discount
     */
    public void setDiscount(double newDiscount) {
        assert newDiscount >= 0;
        this.discount = newDiscount;
    }
}
```

Class 23: ShoppingCart

Week 10

Exercise 10.5

Write an interface to represent a spellchecker with a method `boolean isWord(String word)` that returns `true` if the given word is spelled correctly. Implement three classes for three different languages.

- Each language implementation should recognize at least three words.
- The spellchecker must correctly handle both uppercase and lowercase letters.
- Add a `main` method that takes two command-line arguments: a language name and a string. The program should split the string into words and print any misspelled words.

```java
interface Spellchecker {
    boolean isWord(String word);
}
```

Interface 1: Spellchecker

```java
import java.util.Arrays;
import java.util.HashSet;

class EnglishSpellchecker implements Spellchecker {
    public HashSet<String> words = new HashSet<String>(
            Arrays.asList("buffalo", "I", "the", "a", "old", "young", "man", "woman",
"boat"));

    public boolean isWord(String word) {
        if (words.contains(word))
            return true;
        var lowercasedFirst = Character.toLowerCase(word.charAt(0)) +
word.substring(1);
        if (words.contains(lowercasedFirst))
            return true;
        return false;
    }
}
```

Class 24: EnglishSpellchecker

```java
import java.util.Arrays;
import java.util.HashSet;

class CzechSpellchecker implements Spellchecker {
    public HashSet<String> words = new HashSet<String>(
            Arrays.asList("tři", "tisíce", "sta", "třicet", "stříbrných",
"stříkaček", "stříkalo", "přes", "střech"));

    public boolean isWord(String word) {
        if (words.contains(word))
            return true;
        var lowercasedFirst = Character.toLowerCase(word.charAt(0)) +
word.substring(1);
        if (words.contains(lowercasedFirst))
            return true;
        return false;
    }
}
```

Class 25: CzechSpellchecker

```java
import java.util.Arrays;
import java.util.HashSet;

class JavaSpellchecker implements Spellchecker {
    public HashSet<String> words = new HashSet<String>(Arrays.asList("I", "Java",
"Factory", "Extended"));

    public boolean isWord(String word) {
        // we could use slices but they copy it char for char anyway
        var buf = "";
        for (var i = 0; i < word.length(); i++) {
            var ch = word.charAt(i);
            if (Character.isUpperCase(ch)) {
                if (!buf.isEmpty() && !words.contains(buf))
                    return false;
                buf = "";
            }
            buf += ch;
        }
        if (!buf.isEmpty() && !words.contains(buf))
            return false;
        return true;
    };
}
```

Class 26: JavaSpellchecker

```java
import java.util.Arrays;

public class SpellcheckerProgram {
    public static void main(String[] args) {
        if (args.length < 2) {
            assertTests();
            System.out.println("SpellcheckerProgram <lang> <...words>");
            return;
        }
        var lang = args[0];
        var otherArgs = Arrays.copyOfRange(args, 1, args.length);
        var text = String.join(" ", otherArgs);
        // in real / more advanced spell checker, this would be done per language
        // capital letters aren't checked properly as the implementations don't know
if
        // the word starts a sentence or not
        var words = splitWords(text);

        var spellchecker = getSpellchecker(lang);

        var failed = false;
        for (var word : words) {
            if (!spellchecker.isWord(word)) {
                failed = true;
                System.out.println(word);
            }
        }

        if (failed) {
            System.exit(1);
        }
    }

    static String[] splitWords(String text) {
        return text.split("[\\s.!?'\"]");
    }

    static Spellchecker getSpellchecker(String lang) {
        return switch (lang) {
            case "cs" -> new CzechSpellchecker();
            case "en" -> new EnglishSpellchecker();
            case "java" -> new JavaSpellchecker();
            default -> throw new Error("Unknown language");
        };
    }

    static boolean checkAllWords(Spellchecker spellchecker, String text) {
        for (var word : splitWords(text)) {
            if (!spellchecker.isWord(word)) {
                return false;
            }
        }
        return true;
    }

    static void assertTests() {
        var cs = getSpellchecker("cs");
        var en = getSpellchecker("en");
        var java = getSpellchecker("java");

        assert checkAllWords(cs,
                "tři tisíce tři třicet tři stříbrných stříkaček stříkalo přes tři
tisíce tři třicet tři stříbrných střech.");
        assert !checkAllWords(cs, "třitisíce");
        assert checkAllWords(en, "Buffalo buffalo Buffalo buffalo buffalo buffalo
Buffalo buffalo");
        assert !checkAllWords(en, "Buffal buffal Buffal buffal buffal buffal Buffal
buffal");
        assert checkAllWords(java, "IExtendedJavaFactoryFactory
FactoryJavaFactoryExtended");
        assert !checkAllWords(java, "IExtendedJavaFactoryFactor");
    }
}
```

Class 27: SpellcheckerProgram

Exercise 10.6

Write an interface to represent a vehicle with a method `int getRemainingRange()` that returns the number of kilometers the vehicle can drive with its current fuel. Implement two classes: a gasoline car and a hybrid car.

- The gasoline car should store the amount of fuel left and its mileage (km per liter).
- The hybrid car should store both the amount of gasoline and electric energy left, along with the mileage for running on gasoline and electricity.
- The hybrid car's `getRemainingRange()` method should compute the total range by considering both gasoline and energy.
- Add an `int drive(int kms)` method that simulates driving the specified distance, depletes the fuel accordingly, and returns the actual number of kilometers driven. For the hybrid car, electricity is used before gasoline.
- Write a `main` method to test both vehicle implementations.

```java
interface Vehicle {
    int getRemainingRange();
    int drive(int kms);

    public static void main(String[] args) {
        // 100km / 5l
        var gasCar = new GasolineCar(100. / 5.);
        gasCar.fuel = 10; // 200km fuel
        // 100km / 5l; 10 km / "energy unit"
        var hybridCar = new HybridCar(100. / 5., 10.);
        hybridCar.fuel = 5; // 100km fuel
        hybridCar.electricalEnergy = 5; // 50km eletricity

        assert gasCar.getRemainingRange() == 200;
        assert gasCar.drive(100) == 100;
        assert gasCar.getRemainingRange() == 100;
        assert gasCar.drive(200) == 100;
        assert gasCar.getRemainingRange() == 0;

        assert hybridCar.getRemainingRange() == 150;
        assert hybridCar.drive(50) == 50;
        assert hybridCar.electricalEnergy == 0;
        assert hybridCar.fuel == 5;
        assert hybridCar.drive(200) == 100;
        assert hybridCar.getRemainingRange() == 0;
    }
}
```

Interface 2: Vehicle

```java
public class GasolineCar implements Vehicle {
    /** liters */
    double fuel;
    /** km per liter */
    final double mileage;

    /**
     * @param mileage km per liter
     */
    GasolineCar(double mileage) {
        this.mileage = mileage;
    }

    @Override
    public int getRemainingRange() {
        return (int)Math.floor(fuel * mileage);
    }

    @Override
    public int drive(int kms) {
        var range = getRemainingRange();
        var toDrive = Math.min(range, kms);
        // l = 1/(km/l) * km
        var consumption = 1./mileage * toDrive;
        fuel -= consumption;

        return toDrive;
    }
}
```

Class 28: GasolineCar

```java
public class HybridCar implements Vehicle {
    /** liters */
    double fuel;
    /** km per liter */
    final double fuelMileage;
    double electricalEnergy;
    /** km per el energy point */
    final double electricityMileage;

    HybridCar(double fuelMileage, double electricityMileage) {
        this.fuelMileage = fuelMileage;
        this.electricityMileage = electricityMileage;
    }

    @Override
    public int getRemainingRange() {
        return (int)(fuel * fuelMileage + electricalEnergy * electricityMileage);
    }

    @Override
    public int drive(int kms) {
        // km
        var range = getRemainingRange();
        // km
        var toDrive = Math.min(kms, range);
        // km
        var rangeEl = (int)(electricalEnergy * electricityMileage);
        // km
        var toDriveEl = Math.min(toDrive, rangeEl);
        // km
        var toDriveGas = toDrive - toDriveEl;
        // l = 1/(km/l) * km
        var consumptionGas = 1./fuelMileage * toDriveGas;
        var consumptionEl = 1./electricityMileage * toDriveEl;
        fuel -= consumptionGas;
        electricalEnergy -= consumptionEl;
        return toDrive;
    }
}
```

Class 29: HybridCar

Exercise 10.7

Write a class `Person` to represent a person with a name and an age. Create two subclasses: `Employee` (which extends `Person`) and `Manager` (which extends `Employee`).

- The `Person` class should have a constructor that takes a name and an age.
- The `Employee` class should add a job title and a salary, with an appropriate constructor.
- The `Manager` class should add a monthly bonus field, with an appropriate constructor.
- Add appropriate getter methods for all fields in each class.
- Add a `getSalary()` method. Make sure `Manager` class takes the manager's monthly bonus into account.
- Write a `main` method to test the inheritance hierarchy by creating instances of each class.

```java
public class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public static void main(String[] args) {
        var person = new Person("P1", 20);
        var employee = new Employee("P2", 40, "Senior OOP Architect", 100_000);
        var manager = new Manager("P3", 40, "Product Manager", 130_000, 20_000);

        assert person.getAge() == person.age;
        assert employee.getAge() == employee.age;
        assert manager.getBaseSalary() == manager.salary;
        assert manager.getSalary() > manager.getBaseSalary();
        assert employee.getBaseSalary() == employee.getSalary();
        assert manager.getJobTitle().equals("Product Manager");
    }
}
```

Class 30: Person

```java
public class Employee extends Person {
    String jobTitle;
    int salary;

    public Employee(String name, int age, String jobTitle, int salary) {
        super(name, age);
        this.jobTitle = jobTitle;
        this.salary = salary;
    }

    public String getJobTitle() {
        return jobTitle;
    }

    public int getSalary() {
        return salary;
    }

    public int getBaseSalary() {
        return salary;
    }
}
```

Class 31: Employee

```java
public class Manager extends Employee {
    int monthlyBonus;

    public Manager(String name, int age, String jobTitle, int salary, int
monthlyBonus) {
        super(name, age, jobTitle, salary);
        this.monthlyBonus = monthlyBonus;
    }

    public int getMonthlyBonus() {
        return monthlyBonus;
    }

    public int getSalary() {
        return salary + monthlyBonus;
    }
}
```

Class 32: Manager

Week 11

Exercise 1.3.45

Chaos. Write a program to study the following simple model for population growth, which might be applied to study fish in a pond, bacteria in a test tube, or any of a host of similar situations. We suppose that the population ranges from 0 (extinct) to 1 (maximum population that can be sustained). If the population at time t is x, then we suppose the population at time $t + 1$ to be $rx(1 - x)$, where the argument r, known as the *fecundity parameter*, controls the rate of growth. Start with a small population—say, $x = 0.01$—and study the result of iterating the model, for various values of r. For which values of r does the population stabilize at $x = 1 - \frac{1}{r}$? Can you say anything about the population when r is 3.5? 3.8? 5?

3.5	3.8	5	2
Oscillation between 4 values	Sustained chaos	Dies from overpopulation in 5 ticks	Sustained $1 - \frac{1}{r}$

```
import common.StdDraw;

class Chaos {
    public static void main(String[] args) {
        var x = 0.01;
        var r = Double.parseDouble(args[0]);
        var buffer = new double[200];
        buffer[0] = x;
        StdDraw.enableDoubleBuffering();

        var t = 0;
        while(true) {
            System.out.format("t = %3d; x = %s\n", t, x);
            x = Math.clamp(nextGeneration(x, r), 0, 1);

            var offset = t % buffer.length;
            if(offset == 0) {
                StdDraw.clear();
                buffer = new double[200];
            }
            buffer[offset] = x;
            for(var toDraw = 0; toDraw < buffer.length; toDraw++) {
                var to = buffer[toDraw];
                StdDraw.line(toDraw / (double)buffer.length, 0, toDraw / (double)
buffer.length, to);
            }
            if(offset % 10 == 0) {
                StdDraw.show();
                StdDraw.pause(100);
            }
            if(offset >= 1 && buffer[offset] == buffer[offset - 1]) { // stabilized
value
                StdDraw.pause(5000);
            }
            t += 1;
        }
    }

    static double nextGeneration(double x, double r) {
        return r * x * (1 - x);
    }
}
```

Class 33: Chaos

Exercise 1.4.26

Music shuffling. You set your music player to shuffle mode. It plays each of the n songs before repeating any. Write a program to estimate the likelihood that you will not hear any sequential pair of songs (that is, song 3 does not follow song 2, song 10 does not follow song 9, and so on).

```
import common.StdRandom;

public class RandomSequenceChange {
    public static void main(String[] args) {
        var arrayLength = 5;
        var rounds = 100;
        var data = new int[arrayLength];
        for(var i = 0; i < data.length; i++)
            data[i] = i;
        // number of times any repetition was found after a random shuffle
        var total = 0;
        for(var i = 0; i < rounds; i++) {
            StdRandom.shuffle(data);
            total += (countSequences(data) > 0) ? 1 : 0;
        }
        var chance = 1 - (total / (double) rounds);
        System.out.printf("The chance of no repetition is %s%%\n", chance * 100);
    }

    static int countSequences(int[] data) {
        var output = 0;
        for(var i = 1; i < data.length; i++)
            if(data[i] == data[i - 1] + 1) output++;
        return output;
    }
}
```

Class 34: RandomSequenceChange

Exercise 1.5.32

Clock. Write a program that displays an animation of the second, minute, and hour hands of an analog clock. Use the method `StdDraw.pause(1000)` to update the display roughly once per second.

```
import common.StdDraw;

public class Clock {
    public static void main(String[] args) {
        StdDraw.setScale(-1, 1);
        StdDraw.enableDoubleBuffering();

        while(true) {
            var time = System.currentTimeMillis();
            var ms = (int) (time % 1000);
            // this could be changed to doubles to have smooth movement
            var seconds = (int) (time / 1000 % 60);
            var minutes = (int) (time / 60_000 % 60);
            var hours = (int) (time / 3_600_000 % 12);
            var secondsAngle = (double) seconds / 60 * 2*Math.PI;
            var minutesAngle = (double) minutes / 60 * 2*Math.PI;
            var hoursAngle = (double) hours / 12 * 2*Math.PI;

            StdDraw.clear();
            StdDraw.setPenColor(StdDraw.BLACK);
            StdDraw.setPenRadius(0.004);
            StdDraw.line(0, 0, Math.sin(minutesAngle) * 0.8, Math.cos(minutesAngle) *
0.8);
            StdDraw.line(0, 0, Math.sin(hoursAngle), Math.cos(hoursAngle));

            StdDraw.setPenColor(StdDraw.RED);
            StdDraw.setPenRadius(0.002);
            StdDraw.line(0, 0, Math.sin(secondsAngle), Math.cos(secondsAngle));

            StdDraw.setPenRadius(0.004);
            StdDraw.setPenColor(StdDraw.BLACK);
            StdDraw.circle(0, 0, 1);

            StdDraw.show();
            StdDraw.pause(1000 - ms);
        }

    }
}
```

Class 35: Clock

Week 12

Exercise 12.07

Write a class `Pair` with two type parameters `A` and `B` to represent an immutable pair of values (i.e. the class should have two `final` fields of type `A` and `B`).

- Add an appropriate constructor and getter methods.
  - Do not add any setters, as the class should be immutable.
- Add a method `swap` to the `Pair` class. The `swap` method should return a **new** pair where the first component becomes the second component and vice versa. For example, for the pair `(true, 42)` the method should return `(42, true)`.
- Add methods `withFst` and `withSnd` to the Pair class. Each method should take a type parameter `C` and return a new pair where the appropriate component has been updated. For example, calling `withFst` with the integer `42` on the pair `(true, "Hello World")` should return `(42, "Hello World")`.

Exercise 12.08

Write a class `Dict` that takes two type parameters `K` and `V` to represent a dictionary, i.e. a mapping from items of type `K` to items of type `V`. Internally, the dictionary should maintain a single array of pairs of type `Pair<K, V>`. Add the following methods:

- `V get(K key)` returns the value associated with the given key, or null if the key is not found.
- `void put(K key, V value)` updates the dictionary with a mapping from the key to the value. If the key already exists, its value is updated. Otherwise, a new pair is added.

You may assume the `Dict` can contain at most 100 entries.

Exercise 12.16

Write a class, which takes one type parameter `E`, to represent a multiset. A multiset is a set that counts how many times it contains each of its elements. Add the following methods:

- `int count(E e)` returns the number of times the element `e` occurs in the multiset.
- `void add(E e)` adds the element `e` to the multiset. (Adding increments its count.)
- `void remove(E e)` removes the element `e` from the multiset. (Removing decrements its count.)
- `int size()` returns the number of different elements in the set (non-duplicate count).

An element can never occur a negative number of times in a multiset.

Hint: Use an internal map of type `Map<E, Integer>`.

Week 13

Exercise 13.01

Explain - in your own words - what is an exception?

An object containing information about why a semi-hidden branching happened, usually used for error states. Thrown exceptions bubble up the call stack, as if a return was used, until there's a registered exception handler for a given code region, usually by try/catch.

Exercise 13.05

For each of the following exceptions, mark whether it is a checked or unchecked:

- `NullPointerException` - unchecked
- `IOException` - checked
- `IllegalArgumentException` - unchecked
- `ArrayIndexOutOfBoundsException` - unchecked
- `NumberFormatException` - unchecked
- `ConcurrentModificationException` - unchecked
- `InterruptedException` - checked

How many of these have you experienced?

Most of these, except the last two as I haven't written threaded programs in java yet.

Exercise 13.07

Write a class to represent a gearbox with five gears and a gear for reverse. Add a method `changeGear(int gear)` to change the current gear. The method must `throw IllegalArgumentException` if the gear is not one of `-1`, `1`, `2`, `3`, `4`, and `5`. Here reverse is represented as `-1`. Write a `class IllegalGearChangeException`, which `extends RuntimeException`, and throw this exception:

a. when switching from any gear other than the first gear into reverse (and vice versa), and b. when skipping one or more gears. For example, it is illegal to switch directly from the first gear to the third gear. It is also not allowed to switch directly from reverse to the fourth gear.

```
package week13;

public class Gearbox {
  private int gear = 1;

  public static class IllegalGearChangeException extends RuntimeException {
  }
  void changeGear(int newGear) {
    if (newGear == gear)
      return;
    if (newGear != -1 && (newGear < 1 || newGear > 5))
      throw new IllegalArgumentException();
    // newGear is now either -1 or 1..=5
    var gearToCheck = newGear;
    if (gearToCheck == -1)
      gearToCheck++;
    if (Math.abs(gear - gearToCheck) != 1)
      throw new IllegalGearChangeException();
    gear = newGear;
  }
}
```

Class 36: Gearbox

Exercise 13.09

Write a class to represent a printer from hell. The class should have a single method `print()`. Whenever this method is called, the printer randomly throws one of the following exceptions: `OutOfPaperException`, `OutOfTonerException`, `PaperJamException`. Write classes for these exceptions. Write a main method, which calls `print()`, catches any exception, prompts the user to take action (e.g. "replace toner"), waits for confirmation from the user, and then calls `print()` again. Bonus points for infuriating or vaguely worded instructions.

```
package week13;

import common.StdRandom;

public class Printer {
  static class OutOfPaperException extends Exception {
    @Override
    public String getMessage() {
      return "No Paper pack found in printer.\n" +
          "Check that there's enough paper left for this print job, that the Paper
pack is clean and that the paper pack chip is undamaged. "
          +
          "Please note that third-party Paper packs may not be supported. Check that
the Paper pack is made for this model.";
    }
  }

  static class OutOfTonerException extends Exception {
    @Override
    public String getMessage() {
      return "Printing supply pack is empty or was not found.\n" +
          "Check that an official Printing supply pack is correctly inserted in the
printer, try re-seating if necessary. "
          +
          "Check that Printing supply packs are inserted into the correct color coded
slots. Check that your Printing resupply subscription is active. "
          +
          "Please note that third-party Printing supply packs may not be supported.
Check that the Paper pack is made for this model.";
    }
  }

  static class PaperJamException extends Exception {
    @Override
    public String getMessage() {
      return "lp0 on fire.";
    }
  }

  private static Exception getExceptionFromErrorCode(int errorCode) {
    return switch (errorCode / 10) {
      case 0 -> new OutOfPaperException();
      case 1 -> new OutOfTonerException();
      case 2 -> new PaperJamException();
      default -> throw new IllegalArgumentException();
    };
  }

  public static void print() throws Exception {
    var rnd = StdRandom.uniformInt(30);
    var exception = getExceptionFromErrorCode(rnd);
    throw exception;
  }

  public static void main(String[] args) {
    while(true){
      try {
        print();
        System.out.println("Printed successfully, thanks for your patronage.");
        break;
      } catch (Exception e) {
        System.err.println(e.getMessage());
        try {
          Thread.sleep(1000);
        } catch(Exception interrupt) {}
      }
    }
  }
}
```

Class 37: Printer

Week 14

Write a graphical user interface for a simple calculator application.

Here are some steps to get started:

- Add a label to show the currently entered number.
- Use horizontal and vertical boxes, or a grid, to construct a layout of buttons numbered from 0 to 9.
- Add buttons for addition, subtraction, multiplication, and division.

The calculator should work as follows: You press "5", it shows up in the display, then you press "+", and then you press "7" and it becomes "12".

Hint: Experiment with the calculator on your system to discover how it works.

Here are some optional extensions to consider:

- Style the label to make the result bigger.
- Style the digit buttons so they are bigger.
- Add an event handler such that user can press the keys 0 to 9 on the keyboard with the same effect as using the buttons.
- Add a try-catch block to detect division by zero and show an appropriate alert message.
- Add an event handler such that the escape key resets the calculator.

```
package week14;

import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

/*
This could have been implemented with a possibly more readable state machine instead
of this messy distributed state..

Value => result of operations, or the first number pressed in before selecting an
operation
Buffer => currently typed number, right side of operations when activated
Number pressed => buffer is !empty / was interacted with
Buferred operation => operation selected but not acted on yet
Prev buffer => previously used buffer, in case the user wants to repeat operations
(by pressing = multiple times)
Prev operation => last used operation

Display => text output (reactive string property)
*/
public class Calculator extends Application {
  enum Operation {
    ADD,
    SUBSTRACT,
    MULTIPLY,
    DIVIDE;

    public String toString() {
      return switch(this) {
        case ADD -> "+";
        case SUBSTRACT -> "-";
        case MULTIPLY -> "*";
        case DIVIDE -> "/";
        default -> "";
      };
    }

    public static Operation fromString(String op) {
      return switch(op) {
        case "+" -> ADD;
        case "-" -> SUBSTRACT;
        case "*" -> MULTIPLY;
        case "/" -> DIVIDE;
        default -> null;
      };
    }

    public double apply(double left, double right) {
      return switch(this) {
        case ADD -> left + right;
        case SUBSTRACT -> left - right;
        case MULTIPLY -> left * right;
        case DIVIDE -> left / right;
      };
    }
  }

  double value = 0;
  double buffer = 0;
  boolean numberPressed = false;

  Operation bufferredOperation = null;

  double prevBuffer = 0;
  Operation prevOperation = null;

  StringProperty display = new SimpleStringProperty();

  /**
   * Event handler after a number was pressed
   */
  void processNumberInput(int num) {
    assert num >= 0 && num <= 9;
    resetNumberIfNeeded();
    numberPressed = true;
    buffer *= 10;
    buffer += num;

    updateDisplay();
  }

  /**
   * Event handler after backspace was pressed
   */
  void backspace() {
    resetNumberIfNeeded();
    var num = buffer % 10;
    buffer -= num;
    buffer /= 10;

    updateDisplay();
  }

  /**
   * Applies operations, used as event handler after = / enter
   */
  void processBuffered() {
    if(numberPressed && bufferredOperation != null) {
      value = bufferredOperation.apply(value, buffer);
      prevOperation = bufferredOperation;
      bufferredOperation = null;
      resetBuffer();
    } else if(prevOperation != null) {
      value = prevOperation.apply(value, prevBuffer);
    } else if(bufferredOperation != null) {
      prevBuffer = value;
      value = bufferredOperation.apply(value, value);
      prevOperation = bufferredOperation;
      bufferredOperation = null;
    }
    updateDisplay();
  }

  /**
   * Selects an operation, computes previous operation if still buffered
   */
  void processOperationInput(Operation op) {
    assert op != null;
    if(numberPressed && bufferredOperation != null) {
      value = bufferredOperation.apply(value, buffer);
      prevOperation = bufferredOperation;
      bufferredOperation = null;
    } else if(numberPressed) {
      value = buffer;
    }
    bufferredOperation = op;
    resetBuffer();
    updateDisplay();
  }

  /**
   * Resets all relevant state
   */
  void reset() {
    resetBuffer();
    value = 0;
    bufferredOperation = null;
    prevBuffer = 0;
    prevOperation = null;
    updateDisplay();
  }

  /**
   * Makes sure value is safe to interact with (enter numbers over)
   * resets to 0 if NaN or infinite
   */
  void resetNumberIfNeeded() {
    if(!Double.isFinite(value)) value = 0;
  }

  void updateDisplay() {
    if(numberPressed) display.set(Double.toString(buffer));
    else if(Double.isNaN(value)) display.set("ERROR");
    else display.set(Double.toString(value));

    // System.out.format("value %s buffer %s numberPressed %s bufferredOperation %s
prevBuffer %s prevOperation %s\n", value, buffer, numberPressed, bufferredOperation,
prevBuffer, prevOperation);
  }

  void resetBuffer() {
    prevBuffer = buffer;
    buffer = 0;
    numberPressed = false;
  }

  /**
   * Builds the ui
   *
   * VALUE/BUFFER/ERROR
   *
   * 7 8 9 + =
   * 4 5 6 -
   * 1 2 3 *
   *   0   / =
   */
  public void start(Stage primaryStage) {
    updateDisplay();
    var output = new Label();
    output.setStyle("-fx-font-size: 30px; -fx-text-alignment: right;");
    output.textProperty().bind(display);

    var numberPanel = new GridPane();

    for(var i = 0; i < 9; i++) {
      var button = new Button(Integer.toString(i + 1));
      var index = i;
      button.setOnMouseClicked(e -> {
        processNumberInput(index + 1);
      });
      numberPanel.add(button, i % 3, 2-(i / 3));
    }
    var zero = new Button("0");
    zero.setOnMouseClicked(e -> {
      processNumberInput(0);
    });
    numberPanel.add(zero, 1, 3);

    var controlPanel = new GridPane();

    var values = Operation.values();
    for(var i = 0; i < values.length; i++) {
      var op = values[i];
      var button = new Button(op.toString());
      button.setOnMouseClicked(e -> {
        processOperationInput(op);
      });
      controlPanel.add(button, 0, i);
    }

    var backspace = new Button("⌫");
    backspace.setOnMouseClicked(e -> backspace());
    controlPanel.add(backspace, 1, 0);
    var equals = new Button("=");
    equals.setOnMouseClicked(e -> processBuffered());
    controlPanel.add(equals, 1, 5);


    var hbox = new HBox();
    hbox.setAlignment(Pos.CENTER);
    hbox.getChildren().add(numberPanel);
    hbox.getChildren().add(controlPanel);

    var vbox = new VBox();
    vbox.setAlignment(Pos.TOP_CENTER);
    vbox.getChildren().add(output);
    vbox.getChildren().add(hbox);
    VBox.setVgrow(hbox, Priority.SOMETIMES);

    Scene scene = new Scene(vbox, 300, 250);
    scene.setOnKeyPressed(ev -> {
      final var ch = ev.getText();
      try {
        var num = Integer.parseInt(ch);
        processNumberInput(num);
        return;
      } catch(NumberFormatException e) {}
      var op = Operation.fromString(ch);
      if(op != null) {
        processOperationInput(op);
        return;
      }
      if(ev.getCode() == KeyCode.BACK_SPACE) {
        backspace();
        return;
      }
      if(ev.getCode() == KeyCode.ESCAPE) {
        reset();
        return;
      }
      if(ev.getCode() == KeyCode.ENTER || ev.getCode() == KeyCode.EQUALS) {
        processBuffered();
        return;
      }
    });
    primaryStage.setTitle("Calculator");
    primaryStage.setScene(scene);
    primaryStage.show();
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```