Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme

Prof. Dr. Reinhard von Hanxleden
Christoph Daniel Schulze, Steven Smyth

Institut für Informatik

6. Übung zur Vorlesung „Programmierung“

Sixth Sense: More Classes
Abgabe am Sonntag, 11. Dezember 2016 - 23:55

This week, we will continue our endeavours to write classes that do things. Note that we're starting to put classes into packages. In each assignment, we tell you which package to put your classes in (or the package is clear from the source code you download).

Also note that we of course expect you to write Javadoc comments for every class entry, even if this is not explicitly mentioned in an assignment!

Aufgabe 1 - A (Mandatory) Programmer's Date — 49 Punkte

Part a) If the misguided preconceptions other students have about computer scientists are to be believed, asking a programmer for a date might not exactly get you what you want. In this case, however, we actually *do* want you to write a `Date` class. It should be placed in a package called `programming.set6.date` and provide the following public entries:

- A static method `getDaysInMonth(int year, int month)` that returns the number of days in the given month of the given year, where `1` represents January and `12` represents December. If `month` is out of range, return `0`.
- A static method `validate(int year, int month, int day)` that checks if the given three integers represent a valid date. Remember that the months differ in how many days they have.
- A constructor that takes the year, the month, and the day to initialize the `Date` instance, each as an integer. The constructor should check if the three numbers specifiy a valid date. If they do not, throw an `IllegalArgumentException` which indicates that the date is not valid. Throwing an exception will immediately terminate the constructor call and your program. Here is the code for throwing that exception:

```java
throw new IllegalArgumentException("This is not a valid date.");
```

- Getter methods `getDay()`, `getMonth()`, and `getYear()`.
- A method `dayOfYear()` that returns the number of the day represented by the `Date` instance. For example, January 20th is the 20th day of the year, while May 20th is the 140th day of the year.
- A method `sameYearDiff(Date other)` that calculates how many days the other `Date` is off and returns that as an integer. If `other` lies in the future, that number is positive; otherwise, it is negative. The method is only properly defined if both `Date` instances represent dates in the same year. Otherwise, return `0`.
- A `toString()` method that returns the date in String form like this: `November 20, 2015`.

Remember that February has 29 days in leap years. A year is a leap year if it is divisible by 4 and not divisible by 100 (unless it is also divisible by 400). We expect you to write complete Javadoc comments for your class as well as for all its entries.

Note: The Java runtime library of course already provides classes that can handle dates. However, we want you to write your own class, without using any of the predefined ones. We want you to learn stuff in the process.

Part b) Explain the following Java keywords and terms and give at least one example for each:

- `super`
- `this`
- `static`
- `null`
- Local variable
- Instance variable
- Class variable
- Constructor

Aufgabe 2 - A (Mandatory) Existential Crisis — 49 Punkte

Not to scare you or anything, but during your career as a programmer, you will often come across errors in your code or, even worse, in other people's code. It is thus a good idea to get used to finding and correcting mistakes. So, here's your chance to learn everything there is about working in teams with all the frustration that goes with it.

We have cobbled together a class called `HandyInt` that you can find here. It simply wraps an integer and provides a few arithmetic operations. Download the file and import it into a project in Eclipse.

The following program uses `HandyInt` to calculate $5 \cdot 2^3$:

```java
package programming.set6.arithmetic;

import acm.program.ConsoleProgram;

public class ComplexMathSolver extends ConsoleProgram {
    @Override
    public void run() {
        HandyInt factor = new HandyInt(5);
        HandyInt exp = new HandyInt(3);

        HandyInt result = multiplyWithPowerOfTwo(factor, exp);

        println(factor + " * 2^" + exp + " = " + result);
    }

    /**
     * Returns the result of multiplying {@code factor} with 2 to the power of
     * {@code exp}.
     *
     * @param factor
     *            the handy integer to multiply with the power.
     * @param exp
     *            the exponent of the power.
     * @return the result of the multiplication.
     */
    private HandyInt multiplyWithPowerOfTwo(HandyInt factor, HandyInt exp) {
        HandyInt one = new HandyInt(1);

        while (exp.getInt() > 0) {
            factor = factor.add(factor);
            exp = exp.sub(one);
        }
        return factor;
    }
}
```

We would expect the following output:

```
5 * 2^3 = 40
```

However, what we get is this:

```
40 * 2^0 = 40
```

Find the problem in the `HandyInt` class and fix it such that `ComplexMathSolver` produces the correct result when using your version of `HandyInt`. You are not allowed to change `ComplexMathSolver` in any way. Be sure to comment your fix and explain why the problem occurred. Submit your version of `HandyInt`.

Hint: You may find it handy to check the current state of objects using `println(...)` calls to check if the objects contain the values you think they contain.

Aufgabe 3 - Something for the Children 1 Punkt

You may remember a time when you were younger. Much younger. In this assignment, you'll be writing a program to teach people as young as you once were (a long time ago) to calculate sums and differences of numbers (in a way, this will teach you everything there is to know about didactics).

Write a class `MathQuiz` that is a subclass of `ConsoleProgram` (because children love text interfaces so much) in the package `programming.set6.quiz`. Your program should behave as follows:

- Print at least one line of welcome text. The exact text is up to you, but try not to insult the user too much.
- Pose a series of five math problems for the user to answer.

Each math problem must be stated by your program like this:

```
What is <operand1> <operation> <operand2>?
```

Here, `operand1` and `operand2` are numbers, and `operation` is either `-` or `+`. Your program should generate the operands and the operation randomly by making use of the `RandomGenerator` class introduced in the lecture. The constraints are that all numbers involved (the two operands as well as the result of the operation) should be between 0 and 20 (inclusive). As an example, `What is 4 + 2?` would be OK, but `What is 40 + 2?` and `What is 2 - 4?` would not (out-of-range operand and out-of-range result, respectively).

Of course, your program will have to figure out whether the result was correct or not. Write a method that does that:

```
/**
 * Checks whether the given result is really the result of the operation on
 * the two operands.
 *
 * @param operand1
 *            the first operand.
 * @param operator
 *            the operator, given as a String. This must be either
 *            {@code "+"} or {@code "-"}. Otherwise, this method will always
```

```java
 *            return {@code false}.
 * @param operand2
 *            the second operand.
 * @param result
 *            the proposed result.
 * @return {@code true} if {@code result} is really the result of the
 *         calculation, {@code false} otherwise.
 */
public boolean isCorrect(int operand1, String operator, int operand2, int result) {
    // Implement this method
}
```

After the user answers a problem, your program should print whether the answer was correct or not. If it was not correct, your program should also mention the correct result. Getting the same line of text after each problem would be boring, however, so write two methods that generate the text to be printed after a problem:

```java
/**
 * Returns a message that can be displayed to the user after successfully
 * solving a problem. There are at least four messages the method randomly
 * chooses from to keep it interesting.
 *
 * @return randomly selected success message.
 */
private String generateSuccessMessage() {
    // Implement this method
}

/**
 * Returns a message that can be displayed to the user after failing to
 * solve a problem correctly. There are at least four messages the method
 * randomly chooses from to keep it interesting. The correct result is
 * included in the message somewhere to teach the user a lesson.
 *
 * @param correctResult
 *            the number that would have been the correct result. This
 *            number is included somehow in the returned messsage.
 * @return randomly selected fail message.
 */
private String generateFailMessage(int correctResult) {
    // Implement this method
}
```

Here's what running the quiz could look like:

```
Hi there, this is a math quiz. Stick around though,
it won't be too bad.
What is 5 + 6? 11
Yay!
What is 4 - 3? 2
Not quite. It's 1.
What is 3 + 3? 6
You did it! I'd be proud of you if I wasn't a computer incapable of being proud of you.
What is 10 + 1? 10
What's wrong with you? 11 would have been correct.
What is 19 - 5? 14
```

```
Not bad, not bad. Not great either, but not bad.
```

Be sure to get the lines where you pose the math problem exactly as they are in the example to make it easy for the tests to parse out the result.

Zusatzaufgabe 4 - Graph Plotting 1 Punkt

This assignment will continue where assignment 4 of set 4 left off and visualize the integration. This is quite a complex task, so we will divide it into several parts that you will work through. While you're working on this assignment, don't just blindly follow the instructions. Instead, try to understand why we've split the task into the different subtasks and methods. There's much to be learned there. Also, for each of them, we recommend that you start by working through the problem by hand first before starting to write code. When you're finished, this is what you'll end up with:

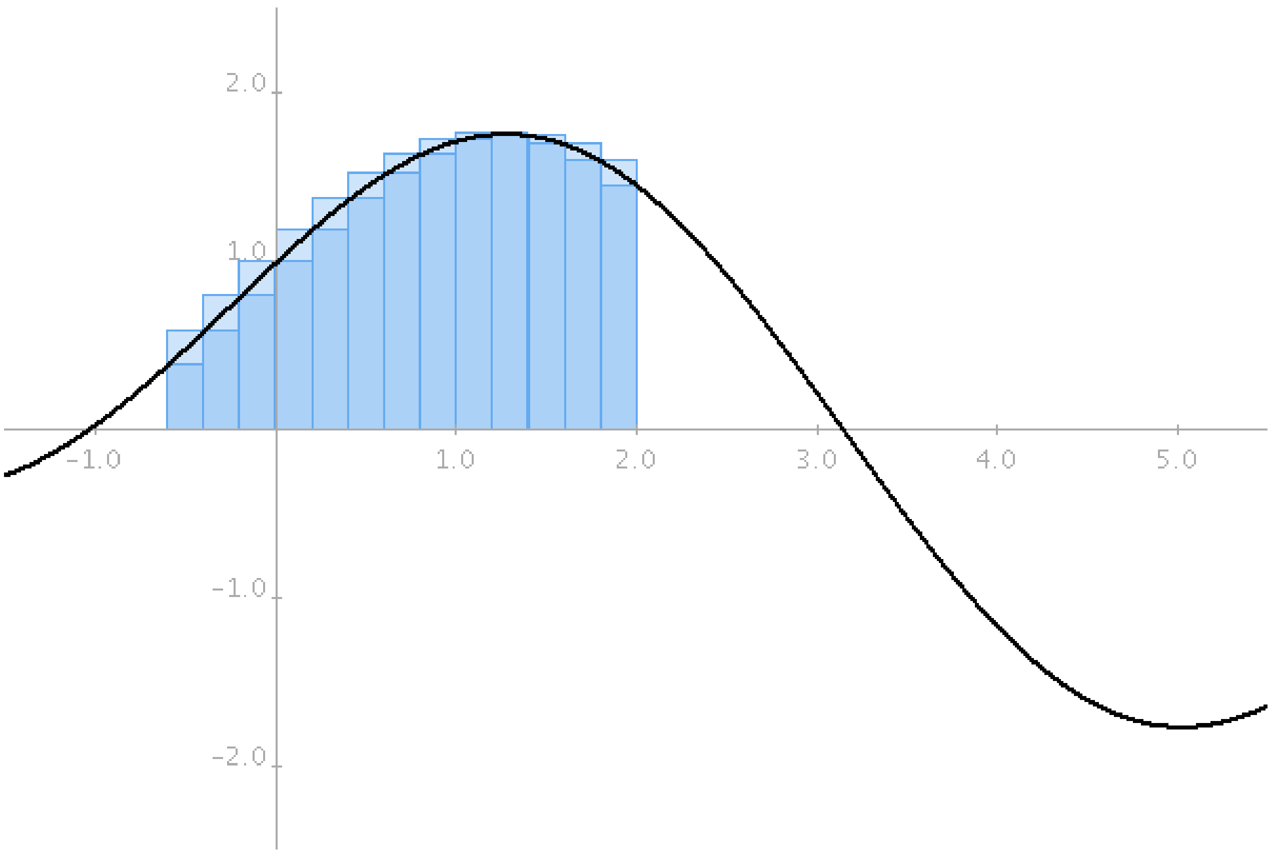

Abbildung 1:

Before we really start, create a new class `GraphPlotter` that is a subclass of `GraphicsProgram` in package `programming.set6.plotter` and paste the following constants into your class:

```java
/** Width of the drawing. */
private static final int WIDTH = 600;
/** Height of the drawing. */
private static final int HEIGHT = 400;
/** Min x coordinate. */
private static final double MIN_X = -1.5;
/** Max x coordinate. */
private static final double MAX_X = 5.5;
```

```java
/** X tick interval. */
private static final double X_TICK_INTERVAL = 1;
/** Min y coordinate. */
private static final double MIN_Y = -2.5;
/** Max y coordinate. */
private static final double MAX_Y = 2.5;
/** Y tick interval. */
private static final double Y_TICK_INTERVAL = 1;
```

Feel free to play around with the constants while you're working on this assignment. However, when uploading your results restore them to the values seen here, as the automatic tests depend on these values.

Next, copy your implementations of the following three methods from assignment 4.4 into your new class:

- `public double f(double x)`
- `public double minRectHeight(double x, double width)`
- `public double maxRectHeight(double x, double width)`

Finally, add a `run()` method. The run method will be empty for now, but as you work through the exercise you will add calls to the methods you write to be able to test your program. We don't really care what your `run()` method looks like, though; the tests will call the other methods directly.

Converting Between Coordinate Systems

The first thing to realize is that the screen coordinate system you're placing your graphical objects in is not the same as the graph's coordinate system. Take, for example, the picture at the beginning of the assignment, which was drawn using the constant values defined above. Using those values, you will see that the origin of the graph's coordinate system, $(0, 0)$, is not in the upper left corner, but actually at screen coordinates $(129, 200)$ (with other constant values the exact screen coordinates would of course be different). Thus, we will need conversion methods that convert screen coordinates to graph coordinates and vice versa. Start by writing those:

```java
/**
 * Calculates the effective x screen coordinate for the given x coordinate
 * in the graph's coordinate system.
 *
 * @param x
 *            the x coordinate.
 * @return the screen coordinate.
 */
public double xToScreen(double x) {
    // Implement this method
}

/**
 * Calculates the coordinate in the coordinate system for the given x
 * coordinate on the screen.
 *
 * @param screenX
 *            screen x coordinate.
 * @return the coordinate system coordinate.
 */
public double screenToX(double screenX) {
    // Implement this method
}
```

```java
/**
 * Calculates the effective y screen coordinate for the given y coordinate
 * in the graph's coordinate system.
 *
 * @param y
 *            the y coordinate.
 * @return the screen coordinate.
 */
public double yToScreen(double y) {
    // Implement this method
}
```

You may wonder why there is no `screenToY(double screenY)` method. The reason is that it is simply not required to solve this assignment.

Drawing the Coordinate System

Now that you have the coordinate conversion methods at hand, it's time to draw the coordinate system:

```java
/**
 * Sets up the x axis of the coordinate system. The method assumes that the
 * coordinate system's top left corner is at (0, 0). Its width and height as
 * well as the tick marks are controlled through the constants defined in
 * this class.
 */
public void drawXAxis() {
    // Implement this method
}

/**
 * Sets up the y axis of the coordinate system. The method assumes that the
 * coordinate system's top left corner is at (0, 0). Its width and height as
 * well as the tick marks are controlled through the constants defined in
 * this class.
 */
public void drawYAxis() {
    // Implement this method
}
```

Here's a few hints to get you going:

- First, check if the axis to be drawn should actually be visible on the screen. If not, don't even bother drawing it. If you draw them, they must span the whole drawing area from `0` to `WIDTH` / `HEIGHT`.
- Draw only the visible tick marks. The tick mark interval specifies where tick marks should be drawn.
- How far your tick marks extend from the axis is not that important, but it's important that they touch the axis. The exact placement of the labels is not that important either, but they should be in the general vicinity of the tick marks. Note that the coordinate system's origin does not have tick marks nor labels.
- We used `new Color(170, 170, 170)` to draw the coordinate system (including the labels), but you're free to choose another color.

Plotting the Graph

Now it's time to actually visualize the graph:

```java
/**
 * Plots the graph by iterating over all x coordinates and generating
 * points. A point is only added if it's in the visible area. Points are
 * visualized through tiny rectangles with a side length of 1.
 */
public void plotGraph() {
    // Implement this method
}
```

To do so, go through all x coordinates on the screen, compute the corresponding function value's y coordinate on the screen and simply place a rectangle there with a side length of 1. There are cleverer ways to draw a graph, but this one is perfectly fine for now.

Drawing the Rectangles

Finally, write code to draw the rectangles used to approximate the area in assignment 5.3:

```java
/**
 * Draws the rectangles used to approximate the graph's area. All inputs are
 * specified in the graph's coordinate system.
 *
 * @param minX
 *            leftmost rectangle coordinate.
 * @param maxX
 *            rightmost rectangle coordinate.
 * @param rectWidth
 *            width of the rectangles.
 * @param minRects
 *            if {@code true}, the min rects are drawn; otherwise, the max
 *            rects are drawn.
 */
public void drawRects(double minX, double maxX, double rectWidth, boolean minRects) {
    // Implement this method
}
```

Here's a few hints:

- Don't draw anything outside the visible area.
- We use `new Color(100, 170, 240, 80)` as the fill color and `new Color(100, 170, 240, 255)` as the border color for the rectangles, but you're free to use other colors.