Homework 2

CS:2230 Computer Science II: Data Structures

Deadline: 3 October 2015 (Saturday) at 11 pm.

Instructions

The contents necessary to complete the assignment, however, will be covered until slightly after the announcement time (but before one week before the deadline). This is done so to motivate attention to lectures and discussions. Rules:

- Submission through ICON, not email, of the source files (**.java**) only.
- Submissions must be of high quality, without errors. This time, comments will be graded.
- The homework is to be completed alone. You can still talk with peers about solution ideas and general concepts.
- Do not use libraries nor additional classes unless permitted by the problems.

For this homework, create a single project. Name it *homework2* or something alike. Download the associated **.zip** file for the homework and uncompress it inside the **src**/ folder of your project. This will create two packages: **edutypes** and **forestfires**. Each one represents a different problem.

1 Types in college

In this problem, you are to implement the following ~~classes~~ **types** in package **edutypes**: **Instructor**, **Professor**, **Student**, **TeachingAssistant**. The subtype relations among these are:

- All classes subtype **Person**, which is **abstract**.

- **Professor** and **TeachingAssistant** subtype **Instructor**.
- **TeachingAssistant** subtypes **Student**.

You are free to use either or both **implements** and **extends** for subtyping. Only **Professor**, **Student**, and **TeachingAssistant** need to be *concrete*.

Implement all these classes in file **Person.java**, which is available in package **edutypes**. Do not implement methods nor fields in these classes.

1.1 Making use of the types

To test your implementation, use the static method **testTypes()** in class **TypesTester**. This method contains the following lines:

```
public static void testTypes() {
  Person a = new Professor(), b=new Student(), c=new TeachingAssistant();
  System.out.println(a);
  System.out.println(b);
  System.out.println(c);
  Instructor d = new Professor(), e=new TeachingAssistant();
  System.out.println(d);
  System.out.println(e);
}
```

After running it, you should get an output such as:

```
edutypes.Professor@139a55
edutypes.Student@1db9742
edutypes.TeachingAssistant@106d69c
edutypes.Professor@52e922
edutypes.TeachingAssistant@25154f
```

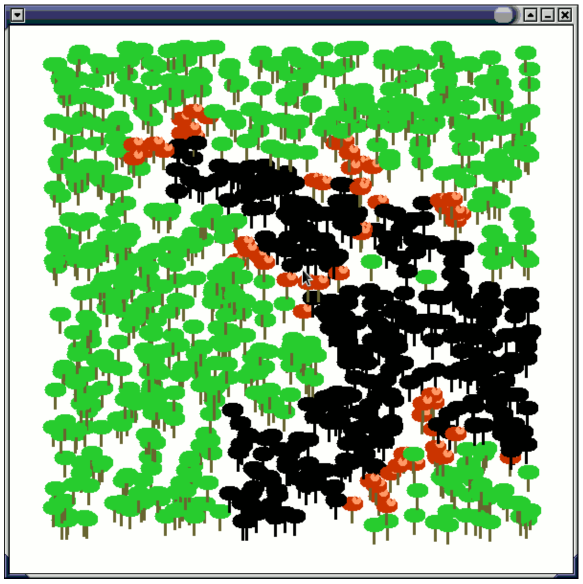
1.2 Submission

Submit only file **Person.java** from package **edutypes**.

2 Simulation of forest fires

Forest fires are a great concern to the general public. The phenomenon of forest fires occurs when one or more trees start burning, and the fire is passed the trees in the proximity. As more trees are involved, the fire becomes uncontrollable.

You are to implement a portion of a *forest fire simulator*. You are provided with some code for this task. You have a visualizer (**FFViewer** class) that is provided to you, and you are also provided some code for testing the implementation of your trees. A screenshot of a finished homework is shown below.



How a simulation in progress looks like.

Simulation model The simulation model we are implementing consists of trees and a distance for spreading the fire. The rules are:

- A tree can be in one of three situations: *healthy*, *burning*, and *burnt*.
- If a *healthy* tree is set on fire, then it is in the *burning* situation.
- The *burning* of a three consists of three days, including the day it was set on fire.
- After the three days of burning, the tree is *burnt*.
- A tree that is burning or burnt cannot acquire fire again. A tree can be set on fire only once.

With respect to the spatial characteristics of the simulation model:

- If a healthy tree is within *d* distance of a burning tree, it will be on fire the next day.

With respect to the simulation process:

- The simulation will continue executing until no trees are burning, i.e., until each tree is either healthy or burnt.

Requirements You are to implement two classes:

- The class for the *tree*. Note that the *internal state* of the tree *changes* over the *days*.
- A class for *custom simulation*. A class for basic simulation is provided. However, you must *extend* this class to make it more flexible.

2.1 Class FFTree

Class **FFTree** must have the methods specified below. You are free to describe the internal state of the class (objects) in any way you want, as long as it consists of primitive variables. Any fields and additional methods must be declared private.

```
package forestfires;

public class FFTree {

  // You choose which fields to use. They must be private.

  // Constructor. It takes in coordinates x and y.
  public FFTree(int x, int y)

  // Returns the x-coordinate of the tree.
  public int getX()

  // Returns the y-coordinate of the tree.
  public int getY()

  // Computes the Euclidean distance (norm 2) between this
  // tree and the other tree
  public double distanceTo(FFTree other)

  // Sets the tree on fire. Has an effect only if the tree is
  // healthy.
  public void setFire()

  // True if and only if the tree is currently on fire.
  public boolean isBurning()

  // True if and only if the fire consumed the tree.
  public boolean isBurnt()

  // Advances one day for the tree.
  public void newDay()

  // Additional methods (if you happen to need them) are to be
  // left private.
}
```

2.2 Class FFSimulator

This class is provided. It reproduces a forest fire. After using it, read its contents, but do not modify it. You need to understand how it works for the next part.

2.3 Class FFCustomSimulator

You must write the code of this class. Instructions:

```
package forestfires;

public class FFCustomSimulator extends FFSimulator{

  // The fields are up to you. They must be left private.

  // Constructor. It does the same as FFSimulator(x,y).
  public FFCustomSimulator(int x, int y)

  // Constructor. It works the same as FFSimulator(500,2).
  public FFCustomSimulator()

  // Constructor. Uses the provided array instead of its own.
  public FFCustomSimulator(FFTree[] aforest)

  // Sets the animation speed. The parameter states how many
  // seconds uses a day in the simulation. The original speed
  // is 0.1 seconds per day.
  // Hint: there is a field in FFSimulator that you can use here.
  public void setFrameSpeed(double seconds)

  // Sets the distance for the spread of fire. Initially, the
  // distance is set to 20 pixels (method spreadFire).
  public void setSpreadDistance(int pixels)
}
```

Warning: you might need to override methods from FFSimulator in this part. However, method **simulation()** does not need overriding.

Hints:

- **FFTester** has some simple code for testing **FFCustomSimulator**.
- In principle, the behavior of **FFCustomSimulator** is identical to that of **FFSimulator**, until you use one of the new methods.

2.4 Class FFTester

This class is provided. Class **FFTester** provides 5 static methods for testing whether your implementation is correct or not.

void case1() It draws three trees diagonally. Total time: 4 seconds.

void case2() It draws some trees. Some are burning. Two seconds later, the burning trees should be burnt. This checks the proper implementation of FFTree with respect to the three-day burning policy. Total time: 4 seconds.

void case3() It runs an actual simulation using **FFSimulator**. This truly evaluates whether **FFTree** is behaving as expected.

void case4() More animations using **FFSimulator**.

void case5() It runs three different simulations using **FFCustomSimulator**.

Since the methods are static, they can be called from your **main** method, for testing.

2.5 Class FFViewer

This class is provided. You do not need to modify nor upload it. It is used by **FFTester** and **FFSimulator** (in the **simulation()** method).

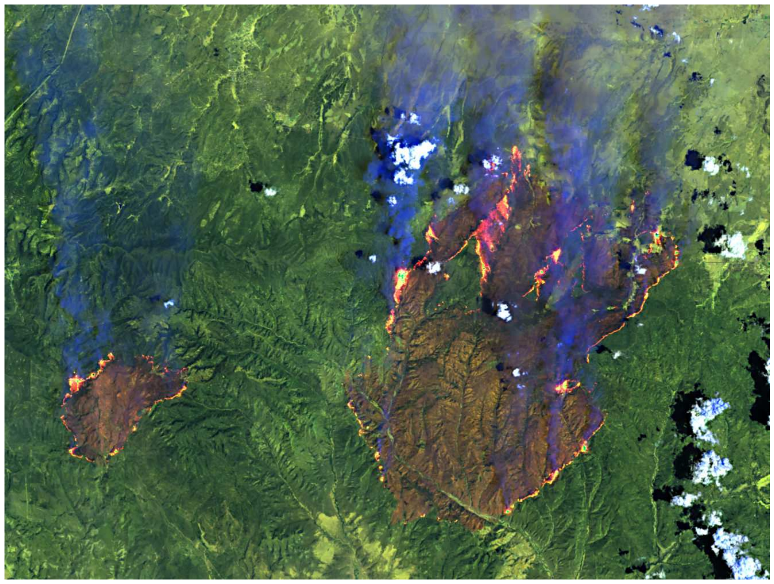
2.6 Submission

Submit only two files: **FFTree.java** and **FFCustomSimulator.java**. The other classes are for your convenience.

2.7 Suggestions

This problem probably demands a fairly amount of programming skill. It teaches about encapsulation and modularity, and a bit of polymorphism. It is meant to increase your general programming skills.

How to start this? Implement a *walking corpe* of **FFTree**, i.e., make it barely functional. Then try **case1** and **case2** of **FFTester**. As **FFTree** becomes more mature, move onto trying it with **case3** and **case4**. After **FFTree** looks complete, move onto **FFCustomSimulator**.



An actual forest fire. Satellite photo by NASA.

Appendix

Class FFViewer

If you are interested in the interface of this class, its methods of interest are listed here:

FFViewer() Public constructor. It does not receive arguments. It creates a new 600x600 window.

void dispose() Destroys the window. If you want the program to end automatically, you must call this method. Otherwise, pressing 'x' on the window is necessary.

void drawTrees(FFTree[] T) Draws an array of trees. If a tree is burning or burnt, you will see this being drawn.

void pause(double seconds) Pauses the viewer (and the application) for **seconds** seconds. For example, make **seconds** equal to 0.1 for $1/10^{th}$ of a second delay. Pausing is necessary for animation. If you draw and don't pause, everything will be too fast for the naked eye!

In the future (homework 4 and 5), you will need to interact directly with the graphical interfaces.