CS1130, Spring 2012. Lecture Sections. Assignment2. JMan

First submission due on the CMS on 28 February at 11:59PM

In this assignment, you will create a simple video game called J*Man. In it, the player uses buttons to steer the star-like J*Man around a board, capturing other pieces. The assignment will give you more experience with the structure of object-oriented programs (e.g. subclasses, casting, and abstract classes). You will develop a few complicated interactive methods. You will need to read and understand the specification of the given code.

To see the game J*Man in action, first download this jar file (which is our solution to the assignment) onto your desktop and then double-click it to start it going. A window will open, with buttons, the game board, and the game rules. Below, we give more extensive rules of the game.

Rules of the Game

Four kinds items appear on the game board.

1. J*Man. J*Man is vaguely asterisk shaped and will be red, yellow, or green. His color determines what other pieces he can capture. There is only one J*Man.
2. White blocks. They just sit there, getting in the way.
3. Walkers. They are triangular. They are red, yellow, or green. As J*Man moves about the board, the walkers wander about slowly.
4. Pillars, which appear as red, yellow, or green disks. Pillars don't move. They change color randomly.

The goal is to use the four navigation buttons to move J*Man about the board until he has captured all the walkers and pillars. J*Man cannot capture a walker or a pillar unless its color is appropriate. J*Man can capture:

- a red piece only if J*Man is green,
- a yellow piece only if J*Man is red,
- a green piece only if J*Man yellow.

Whenever J*Man captures a piece, J*Man takes on that piece's color.

J*Man and the walkers follow the following rule: If they try to move off the board or try to move into a square that is occupied by something they cannot capture (for a walker, any occupied space), they don't move.

After each attempt by J*Man to move, all the Walkers and Pillars also get to perform one action, as follows:

- Approximately 2/3 of the time (this is random), their action consists of doing nothing.
- The other 1/3 of the time, a pillar randomly chooses a color: red, green, or yellow.
- The other 1/3 of the time, a walker attempts to move up, down, left, or right. The choice is random. If the square to move to is off the board or already occupied, the walker doesn't move.

Program structure

The program has two parts:

1. The graphical user interface (GUI), which draws the board in a JFrame window, responds to clicks of buttons, and so forth. GUIs are beyond the scope of this course, and we provide all the GUI code. At the end of this document, we provide some explanation and point you to reading material to learn more about building a GUI in Java. We urge you to study this material, but you don't have to.
2. Code that performs the action of the various pieces. You will write virtually all of this part.

Graphic User Interface (GUI)

We provide the GUI. The button on the top left displays a dialogue that asks the user if they want to start a new game; the button on the right displays the rules of J*Man. The middle of the screen is the game board, and the four buttons on the bottom are used to move J*Man around the board.

Classes

In file a2jmanstart.zip, we provide parts of three classes: `JManGUI`, `Piece`, and `JMan`. Abstract class `Piece` describes the common behavior of the various pieces on the board. You have to complete the bodies of some of its methods and decide what fields it needs. You will write subclasses of `Piece`, whose instances will maintain the pieces —J*Man, blocks, walkers, and pillars. How you do this is up to you, with a few exceptions, one of which is that subclass `JMan` of `Piece` should be used for the instance of J*Man in the game.

The pogram will not compile until you complete the two constructors in class `JMan` because superclass Piece does not have a constructor with no parameters.

Take a look at class `JManGUI`. Much of it has to do with the GUI, and you should focus on

1. Field `board`, which is declared near line 47. This array contains the pieces that are on the board. An array element is **`null`** if that position is empty. The upper left hand corner of the board is (0, 0), i.e. `board`[0][0]. The x-value increases to the right; the y-value increases down.
2. Field `jman`, which contains an object that describes the J*Man.
3. Function `isOnBoard`, near line 212, and the three methods below it. Look at their specs. Then, spend some time understanding their bodies to get a better feel for how field `board` is used. You will be writing calls on these functions.
4. Procedure `putNew`, near line 190. Class `JManGUI` is complete except for the body of this procedure, and you have to complete it. You cannot start testing it until you have at least started the necessary subclasses of class `Piece`. Do not change anything else in class `JManGUI`.
5. Look carefully at procedure `actionPerformed`, near line 300. This procedure is called when a button is pressed. It determines which button was pressed and acts accordingly. If the button was a request to move J*Man in some direction, it sets field `nextJManDirection` to indicate the direction in which J*Man is being asked to move and then calls procedure act().
6. Look carefully at procedure `act`. It first calls `JMan.act` and then searches the board and calls the act procedure of every piece on the board. Notice how it uses `hasActed()` of each piece to make sure it calls procedure `act` only if this piece has not acted yet on this round, and it also sets the has-acted property to true after call procedure `act`. In this way, it makes sure that each piece acts only once per round. This is needed because J*Man and walkers move around the board and hence the loops in procedure act may process these pieces more than once.

Complete the constructors in class JMan

You can start off by completing the constructors in class JMan so that the program compiles. This is fairly easy to do since each can be a call on a super-class constructor.

Completing class Piece

Your first task should be to complete abstract class `Piece`. It is the behavior of the class that is important to the use of the class, and we have defined that behavior by declaring and specifying the methods. You now have to determine what fields are needed in order for the methods to work properly and complete the methods.

Object-oriented programs are often developed in this fashion. One first defines the behavior of a new class by describing its methods, giving the header (return type, name, parameters, etc.) and a careful specification of each method, stubbing in the body just enough so the program will compile. Later, someone (in this case, you) provides the implementation by declaring necessary fields and completing method bodies.

Note carefully the four static fields `BLOCK`, `JMAN`, `WALKER`, and `PILLAR` in class `Piece`. They have been declared with attribute final, which means that they cannot be changed and thus can be considered to be constants. They are used to communicate values to methods.

When such constants have been declared, ALWAYS use them and NOT the values assigned to them.

Note also function `rand` at the end of class `Piece`; call this function to generate random numbers.

Writing the program

Work on the program as follows. This sequence of steps is designed to let you see real progress with each step, giving you confidence that what you are doing is correct.

1. Make putNew create JMan

Fix `JManGUI.putNew` so that it handles (only) the case of parameter `t` being `Piece.JMAN` —read the specification of `putNew` carefully! Now, when you evaluate `new JManGUI()` in the interactions pane, you should see a board with a JMan in its upper left corner (position [0, 0]).

Don't go on to the next step until this part works! In this way, you learn what has to be done as you go and you gain confidence in what you are doing.

2**. Do the following for each of the following kinds of pieces: block, pillar, walker, one at a time.** Don't go on to the next one until the one you are working on works properly!

> Create a class for the new kind of piece with suitable constructor(s), stubbing in any other methods that are needed. Make sure to specify the constructors properly —look at the specs of the constructors in `JMan` as examples. Then fix `JManGUI.putNew` so that it handles parameter `t` being the value for the new kind of piece. When you evaluate `new JManGUI()`, you should see that kind of piece appearing on the board.
>
> Write a function `toString()` in the class —notice that we have written one for you in class `JMan`. This method can be invaluable in debugging; for example, find out about a piece `p` simply by executing `System.out.println(p);`

3. Do the following for each of the kinds of piece; this order may be easiest: block, pillar, walker, J*Man:

> The tendency will be to want to do J*Man first, but that is the hardes act method to write and is best left to the end.
>
> Put in a good specification for procedure `act` (look at the one we give you for `JMan.act` as an example). Write the procedure body. Test it by starting the game and seeing what happens to the pieces when you click an up, down, left, or right button. For walkers and J*Man, be sure you try all possibilities —trying to move off the board in any direction, moving to an empty square, trying to move to an occupied square.
>
> Notice that in `JMan`, we have stubbed in a few methods to help you in implementing `act`.

Important requirement of JMan.act and the Walker.act procedure

In the act procedures for J*Man and walkers, you have to account for moving in one of four directions: up, down, left, and right. You should NOT have essentially duplicate code four times, once for each direction. That leads to a long, unwieldy, difficult-to-debug and maintain method. Instead, your code should (1) calculate the coordinates of the proposed position (px, py) to move to and (2) process the proposed move to (px, py). If you do not write your code this way, you will be required by the grader to change it later.

Summary

We summarize what you need to do in this assignment.

1. Do not make any changes to class`JManGUI` except the body of method `putNew`.

2. Follow the development strategy proposed above, and implement anything you need in order to have the game work properly

3. Before submitting your assignment, click the Javadoc button and check the specification that results. If your specifications on the new classes that you write are not suitable, you will be asked to fix them and resubmit.

5. Test the program any way you want. If your program does not work properly, you will be asked to fix it.

How to submit your assignment

Please put all your .java files for this project in a folder. Then produce a .zip file of this folder, named a2jman.zip. Submit file a2jman.zip on the CMS.

About building GUIs in Java

The best place to learn about building GUIs in Java is to read/listen to pages 17-1 to 17-4 of the CD ProgramLive. You already know that a JFrame is a window on your monitor. Page 17-1 tells you how you can add components to the JFrame, like buttons and text fields.

Page 17-2 then shows you the basic components that one can add to a JFrame, and for each one, you can download the code for the demo.

Page 17-3 then tells you about layout managers, which are used to put components in various positions of a JFrame.

Finally, Page 17-4 shows you the basics of "listening" to actions, like a click of a button.

Probably, in less than 1.5 hours, you can have a good idea about how GUIs are used. It will then be fairly simple to look at class `JManGUI` and see how the GUI is constructed. The GUI is constructed in the constructor. First, the "preferred sizes" of the buttons are defined. Second, the directional buttons are added to a Box. Third, a "listener" is registered for each button, so that the system know which method to call when a button is pressed. Fourth, all the components are added to the JFrame. Fifth, the board is constructed. Sixth, the JFrame is "packed", placed, made unresizable, and shown.

If you understood pages 17-1 to 17-4, you can understand how this GUI is constructed.