Classes, Brackets and Braces Oh My

This is suggested reading before the class starts. It is very technical and you probably will not understand everything, (this is to be expected don't get frustrated) however, since it is difficult to understand it will help you a lot when I explain it again in the class. You are going to be writing code and while most of us think that computers are really smart but they are actually really stupid and they need very precise instructions to do simple things. And you will get stuck by many things that seem trivial as in:

- forgetting a semi-colon **;** at the end of a statement
- having mismatched round brackets/parenthesis **()** or mismatched braces **{}**
- having a semi-colon at the end of a line that shouldn't have one.

Firstly, we have to start with some common constructs in Java.

Watch this video https://youtu.be/eIrMbAQSU34?t=240 starting at the 4 minute mark until it reaches 8:40. (Running time 4 Min 40 seconds). This will give you a quick overview of the anatomy of a Java Program.

Comments

Since we are writing **code**, sometimes it is difficult to understand what is going on in the code and we need to leave ourselves some **notes**. So the way that we do that is called comments. Comments in Java come in 2 flavours.

- Multiline comments
    - These are started by an /* and are terminated by a */ which can span multiple lines
- Ignore to end-of-line comments.
    - These are designated by a // and everything following on the same line will be ignored.

Here are some examples:

```
public void someMethod() {
/*
The above 2 characters start a multi-line comment.
Note that it can span multiple lines until it is terminated
by the next line which contains the special ending 2 characters.
*/
// However, this is a single line comment.
// The next line has a comment after some code
    System.out.println(""); // To the end of the line is a comment
// More code might come after this
}
```

Round Brackets and Braces

There is a rule in Java that for every open bracket or brace you will have a closing bracket or brace. As simple as this sounds, it is easy to forget and you will get caught on this. It takes practice to get this right. If you watched the video above, ignore his comment about putting braces on the same line as the class or function; we are going to always start them on the following line.

Classes and Methods

All code is organized around classes in Java. Within the Greenfoot environment most of our classes will start off looking very similar to this. Pay attention only to the braces and brackets below.

```
public class Rocket extends Actor
{   ①
    public void act()
    {   ②
        // Some more code will go here.
    }   ②

}   ①
```

① Notice that these braces line up - both at column 1 - they define everything within the *class* **Rocket**. This includes the method act().

② Notice that these braces line up - both at column 4 - they define everything within the *method* **act**.

Now, some more explanation is in order. In Greenfoot, there will be a class created for every visual object that we want to create, which are called **actors**. So if we were creating a game where there was a ball that we wanted to hit with a paddle that would bounce around the screen to maybe hit some bricks, we would create a class each for the visual object **Ball**, **Paddle**, and **Brick**. In total - 3 classes. Likewise if we were to design a game in which we want a rocket to shoot bullets at aliens as they come down the screen we would create a class each for the visual object **Rocket**, **Bullet**, and **Alien**. Again coincidentally - 3 classes.

All of these classes would be considered Actors in our game. All actors will have at least one method defined within the class and that is the special method **act**.

Now, more on the Anatomy of Actor classes is in order. So from the example above, we are going to look at the **Rocket** class again. There is a lot going on in here, so we are going to break it down into its pieces.

```
public class Rocket extends Actor ①
{
    public void act()   ②⑦
    {
        if (Greenfoot.isKeyDown("left")) ③
        {   ④
```

```
            //Code here to move the rocket some pixels to the left
        }   ④

        if (Greenfoot.isKeyDown("right")) ⑤
        {
            //Code here to move the rocket some pixels to the right
        }
    }                   ②

    public void shootBullet()   ⑥⑦
    {

    }                     ⑥

}
```

① This is the class definition line. It contains the class name Rocket and that fact that it is a subclass of Actor via the **extends** keyword. Everything in this file now belongs to the class Rocket.

② This is the special method **act**. All actors must implement this method as a minimum. It is defined with the class **Rocket** and the definition of the method starts with its first opening **{** to its corresponding closing **}**.

③ This is an **if** statement. We are checking to see if our player has pushed the **left** arrow key. If they have, we want the **Rocket** to move to the left. Don't worry about how the actual code is doing that for now. However, pay attention to the matching brackets on this line. **Note**: all **if** statements will always have round brackets surrounding everything that we want to test for true or false. In this case, **Greenfoot.isKeyDown("left")** is surrounded by brackets. Look above and verify that that is true.

④ If the above statement is true, all lines between the following **{ }** will be executed. If the above statement is not true, everything between the **{ }** will be skipped.

⑤ Another if statement, similar explanation as to the one above. Note: both of these if statements are within the **act** method.

⑥ This defines another method name **shootBullet** . This method is also defined within the **Rocket** class, but is not within the **act** method since the **act** method is finished its definition on the above line (Closing **}**). It's its own method, presumably where we will code what is going to happen when we want the **Rocket** to shoot a **Bullet**. It is defined to its closing **}**

⑦ Notice that both these methods have an **()** after their name. More on this later.

Again, we will go over all this again in class.