

Started on	Friday, 8 March 2019, 1:45 PM
State	Finished
Completed on	Tuesday, 12 March 2019, 12:46 PM
Time taken	3 days 23 hours
Marks	7.10/8.00
Grade	8.88 out of 10.00 (89%)

Information

General Lab Information

This lab should take you about one week to complete.

Goals:

- Understand how to use local variables, conditionals, and loops.
- Understand how overloading works for methods and constructors.
- Understand how multiple classes are used together.

If you have any issues, have a chat to A or B, your friendly tutors.

Response history			
Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	11/03/19, 15:32	Seen	
3	12/03/19, 12:46	Attempt finished	

Methods and Beyond ...

In the previous lab, we only ever called methods by calling them direct from an object in the main() method. This was to keep things simple, now we're going to look at how to call methods from anywhere.

Firstly, you can call any method of the class you are in by simply writing its name. Let's see an example.

```
public class StarmanSays {

    public void sayDontPanic() {
        System.out.println("Don't Panic!");
    }

    public void sayHelloWorld() {
        System.out.println("Hello, world!");
    }

    public void sayThings() {
        sayHelloWorld();
        sayDontPanic();
    }

    public static void main(String[] args) {
        StarmanSays starman = new StarmanSays();
        starman.sayThings();
    }
}
```

In the sayThings() method, the methods sayHelloWorld() and sayDontPanic() are called. See how they do not have an object in front of them? That means they are called from the current class, and more importantly, the current instance of the class.

But why do some methods have objects in front of them? And why in the main() method do we call methods from the object, surely its the same class right?

Lets consider a different class. Assume you have the following statements in another class:

```
String speech = "I am the great Starman, traveller of the cosmos";
int speechLen = speech.length(); // returns 47
```

In this case, we are calling a method length() that is not within the class, and it also requires explicit information about the instantiated object. **This means we normally call methods on objects to get output that is dependent of the attributes of that object** (the exception are static methods and attributes which do not depend on information of a particular object).

This also brings up a similar reasoning behind why we use object names in main(). When Java calls main(), there are no objects created by default, so we must create an object in order to call non-static methods.

Most classes won't have a main() method. Classes only need a main() method when we are interested as using it as the starting point for our programs. Since all of our example classes so far have been very small, every class has tended to have a main() method.

Response history

Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	11/03/19, 15:32	Seen	
3	12/03/19, 12:46	Attempt finished	

Question **1**

Correct

Mark 1.00 out of 1.00

StarmanFixes Class

Starman is changing a wheel on this Red Tesla Roadster in preparation of his upcoming holiday, and needs to keep track of how many times he has tightened up his wheel nuts. He does it in irregular motions, sometimes turning the socket wrench a quarter turn, a half turn or a full turn. Although sometimes in space, when you are floating in nothingness it can be hard to work out how far you turned, as you might be rotating too.

Implement a class called StarmanFixes with the following:

- a variable called: oneTurn of type double, which holds how far one turn of a socket wrench goes.
- a variable called: tightenAmount of type double, which holds how far the wheel nut has been tightened so far.
- void setOneTurn(double amount), a method which sets the value of oneTurn to amount.
- double getTightenAmount(), a method which returns the value of tightenAmount
- void tightenQuarter(), a method which adds 0.25 of oneTurn to tightenAmount, and then uses System.out.println to output "Starman tightens the nut one quarter of a turn"
- void tightenHalf(), a method which calls tightenQuarter() twice and then uses System.out.println to output "The nut has been tightened half a turn"
- void tightenFull(), a method which calls tightenHalf() twice and then uses System.out.println to output "The nut has been tightened a full turn"

For example:


Test	Result
StarmanFixes wheelnut = new StarmanFixes(); wheelnut.setOneTurn(1); wheelnut.tightenQuarter(); wheelnut.tightenHalf(); wheelnut.tightenFull(); System.out.println(wheelnut.getTightenAmount());	Starman tightens the nut one quarter of a turn Starman tightens the nut one quarter of a turn Starman tightens the nut one quarter of a turn The nut has been tightened half a turn Starman tightens the nut one quarter of a turn Starman tightens the nut one quarter of a turn The nut has been tightened half a turn Starman tightens the nut one quarter of a turn Starman tightens the nut one quarter of a turn The nut has been tightened half a turn The nut has been tightened a full turn 1.75

Answer: (penalty regime: 10, 20, ... %)

```
1 public class StarmanFixes {
2     private double oneTurn = 0.0;
3     private double tightenAmount = 0.0;
4
5     public void setOneTurn(double amount) {
6         oneTurn = amount;
7     }
8     public double getTightenAmount() {
9         return tightenAmount;
10    }
11    public void tightenQuarter() {
12        tightenAmount += 0.25*oneTurn;
13        System.out.println("Starman tightens the nut one quarter of a turn");
14    }
15    public void tightenHalf() {
16        for (int i=0; i<2;i++) {
17            tightenQuarter();
18        }
19        System.out.println("The nut has been tightened half a turn");
20    }
21    public void tightenFull() {
22        for (int i=0; i<2;i++) {
```

	Test	Expected	Got	
--	------	----------	-----	--

	Test	Expected	Got	
✓	<pre> StarmanFixes wheelnut = new StarmanFixes(); \t wheelnut.setOneTurn(1); wheelnut.tightenQuarter(); wheelnut.tightenHalf(); wheelnut.tightenFull(); System.out.println(wheelnut.getTightenAmount()); </pre>	<p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>The nut has been tightened a full turn</p> <p>1.75</p>	<p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>The nut has been tightened a full turn</p> <p>1.75</p>	✓
✓	<pre> StarmanFixes wheelnut = new StarmanFixes(); \t wheelnut.setOneTurn(3.5); wheelnut.tightenHalf(); wheelnut.tightenFull(); wheelnut.tightenFull(); System.out.println(wheelnut.getTightenAmount()); </pre>	<p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>The nut has been tightened a full turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>The nut has been tightened a full turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>The nut has been tightened a full turn</p> <p>8.75</p>	<p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>The nut has been tightened a full turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>The nut has been tightened a full turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>Starman tightens the nut one quarter of a turn</p> <p>The nut has been tightened half a turn</p> <p>The nut has been tightened a full turn</p> <p>8.75</p>	✓

Passed all tests! 

Question author's solution:

```
public class StarmanFixes {

    public double oneTurn;
    public double tightenAmount;

    public void setOneTurn(double amount) {
        oneTurn = amount;
    }

    public double getTightenAmount() {
        return tightenAmount;
    }

    public void tightenQuarter() {
        tightenAmount += 0.25 * oneTurn;
        System.out.println("Starman tightens the nut one quarter of a turn");
    }

    public void tightenHalf() {
        tightenQuarter();
        tightenQuarter();
        System.out.println("The nut has been tightened half a turn");
    }

    public void tightenFull() {
        tightenHalf();
        tightenHalf();
        System.out.println("The nut has been tightened a full turn");
    }

    public static void main(String[] args) {
        StarmanFixes wheelnut = new StarmanFixes();

        wheelnut.setOneTurn(1);
        wheelnut.tightenQuarter();
        wheelnut.tightenHalf();
        wheelnut.tightenFull();
        System.out.println(wheelnut.getTightenAmount());
    }
}
```

Correct
Marks for this submission: 1.00/1.00.

Response history				
Step	Time	Action	State	Marks
1	8/03/19, 13:45	Started	Not complete	
2	11/03/19, 15:32	Submit: public class StarmanFixes { private double oneTurn = 0.0; private double tightenAmount = 0.0; public void setOneTurn(double amount) { oneTurn = amount; } public double getTightenAmount() { return tightenAmount; } public void tightenQuarter() { tightenAmount += 0.25*oneTurn; System.out.println("Starman tightens the nut one quarter of a turn"); } public void tightenHalf() { for (int i=0; i<2;i++) { tightenQuarter(); } System.out.println("The nut has been tightened half a turn"); } public void tightenFull() { for (int i=0; i<2;i++) { tightenHalf(); } System.out.println("The nut has been tightened a full turn"); } }	Correct	1.00
3	12/03/19, 12:46	Attempt finished submitting:	Correct	1.00

Local Variables

When you write larger methods, you are going to need a scratch pad to place values while you are performing calculations, and like other programming languages, Java allows you to have local variables in methods.

They look a little something like this:

```
public class SpaceshipSpeed {
    // These two are class variables / attributes
    private double distance = 0;
    private double time = 0;

    public double calculateSpeed() {
        // currentSpeed is a local variable
        double currentSpeed = distance / time;
        return currentSpeed;
    }

    public static void main(String[] args) {
        // This also makes ship a local variable too
        SpaceshipSpeed ship = new SpaceshipSpeed();

        // distance to Mars (meters)
        ship.distance = 56000000000.0;
        // time starman has been in spaceship (seconds)
        ship.time = 14400;
        System.out.println("Starman averaged " + ship.calculateSpeed() + " m/s to Mars");
    }
}
```

Notice how we did not need to say that the local variable is public? Since the variable is local to the method, it is not visible from outside of the class. Methods can have any number of local variables, so do use them to keep your code tidy and readable.

Response history			
Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	11/03/19, 15:32	Seen	
3	12/03/19, 12:46	Attempt finished	

Method Parameters

You can also pass parameters into methods in order to move data around. When you do this, you need to tell Java the type of the data you are passing in.

For example:

```
public class StarmanSteps {
    public void move(int steps, String travelType) {
        System.out.println("Starman is " + travelType + " " + steps);
    }

    public static void main(String[] args) {
        StarmanSteps starman = new StarmanSteps();
        starman.move(10, "walking");
        starman.move(42, "running");
    }
}
```

This brings us to what we call a method *signature*. A method signature is the combination of the visibility (public / private), the return type (in this example, void), the method name (move), and the list of parameters and their types.

Method signatures are important, because the Java compiler ensures that when a function is called, it conforms to the signature in terms of visibility and the types being passed to it. For example, the compiler would reject 42.2 when passed in as "steps", because it is not an int. Secondly, when a function returns, the compiler checks the signature for the return type. In this example no data is returned, so the type is void. If a String were to be returned instead, the return type must be String.

One of the more strange things about Java is that methods can be *overloaded*. Overloading means having multiple methods of the same name, but do slightly different things based on the number and types of parameters being passed in (so they have different method signatures).

Lets consider the movement example, but this time with some overloaded methods.

```
public class StarmanSteps {
    public void move() {
        System.out.println("Starman walks 1 step");
    }

    public void move(int steps) {
        System.out.println("Starman is walking " + steps + " steps");
    }

    public void move(int steps, String travelType) {
        System.out.println("Starman is " + travelType + " " + steps + " steps");
    }

    public static void main(String[] args) {
        StarmanSteps starman = new StarmanSteps();
        starman.move();
        starman.move(10);
        starman.move(42, "running");
    }
}
```

Here there is one overloaded method, move, and it is overloaded three times each with different signatures:

```
public void move();
public void move(int steps);
public void move(int steps, String travelType);
```

If the methods are called the same, how can Java tell them apart? Java checks the signature to decide what method to call. Note, the number of parameters and the order of parameters is important. Writing

```
starman.move("running");
```

is wrong because there is no method that conforms to the signature

```
public void move(String travelType);
```

Response history			
Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	11/03/19, 15:32	Seen	
3	12/03/19, 12:46	Attempt finished	

Question **2**

Correct

Mark 1.00 out of 1.00

Using Parameters

Starman has found a space station to dock with, and wants to get inside to listen to some music since the stereo in his red Tesla Roadster doesn't work very well in the vacuum of space. There are a few options to choose from: One, to spacewalk into the station, two, to park the car inside a larger airlock and enter, or three, let the station automatically dock the car and start playing a song on the stations entertainment system.

Implement the following overloaded "dock" method in the class SpaceStation:

- void dock(), a method that uses System.out.println() to output "Starman floats toward the space station and enters through the airlock"
- void dock(String vehicle), a method that takes <vehicle> as a String and uses System.out.println to output "Starman enters the large airlock, piloting <vehicle>"
- void dock(String vehicle, String songName), a method that uses System.out.println to output "Starman enters the large airlock, piloting <vehicle>" followed by a newline with: "... and the station cranks up <songName> on the entertainment system"

Note, you could call void dock(String vehicle) from dock(String vehicle, String songName) for some awesome code reuse.

For example:

Test	Result
SpaceStation station = new SpaceStation(); station.dock(); station.dock("Red Tesla Roadster"); station.dock("Red Tesla Roadster", "Space Oddity");	Starman floats toward the space station and enters through the airlock Starman enters the large airlock, piloting Red Tesla Roadster Starman enters the large airlock, piloting Red Tesla Roadster ... and the station cranks up Space Oddity on the entertainment system

Answer: (penalty regime: 10, 20, ... %)

```
1 public class SpaceStation {  
2     public void dock() {  
3         System.out.println("Starman floats toward the space station and enters through the  
4     }  
5     public void dock(String vehicle) {  
6         System.out.println("Starman enters the large airlock, piloting "+vehicle);  
7     }  
8     public void dock(String vehicle, String songName) {  
9         dock(vehicle);  
10        System.out.println("... and the station cranks up "+songName+" on the entertainment  
11    }  
12 }
```

	Test	Expected	Got	
✓	SpaceStation station = new SpaceStation(); station.dock(); station.dock("Red Tesla Roadster"); station.dock("Red Tesla Roadster", "Space Oddity");	Starman floats toward the space station and enters through the airlock Starman enters the large airlock, piloting Red Tesla Roadster Starman enters the large airlock, piloting Red Tesla Roadster ... and the station cranks up Space Oddity on the entertainment system	Starman floats toward the space station and enters through the airlock Starman enters the large airlock, piloting Red Tesla Roadster Starman enters the large airlock, piloting Red Tesla Roadster ... and the station cranks up Space Oddity on the entertainment system	✓

	Test	Expected	Got	
✓	SpaceStation station = new SpaceStation(); station.dock(); station.dock("Falcon Heavy Rocket"); station.dock("Falcon Heavy Rocket", "Rocket Man");	Starman floats toward the space station and enters through the airlock Starman enters the large airlock, piloting Falcon Heavy Rocket Starman enters the large airlock, piloting Falcon Heavy Rocket ... and the station cranks up Rocket Man on the entertainment system	Starman floats toward the space station and enters through the airlock Starman enters the large airlock, piloting Falcon Heavy Rocket Starman enters the large airlock, piloting Falcon Heavy Rocket ... and the station cranks up Rocket Man on the entertainment system	✓

Passed all tests! ✓

Question author's solution:

```
public class SpaceStation {
    public void dock() {
        System.out.println("Starman floats toward the space station and enters through the airlock");
    }

    public void dock(String vehicle) {
        System.out.println("Starman enters the large airlock, piloting " + vehicle);
    }

    public void dock(String vehicle, String songName) {
        dock(vehicle);
        System.out.println("... and the station cranks up " + songName + " on the entertainment
system");
    }

    public static void main(String[] args) {
        SpaceStation station = new SpaceStation();
        station.dock();
        station.dock("Red Tesla Roadster");
        station.dock("Red Tesla Roadster", "Space Oddity");
    }
}
```

Correct
Marks for this submission: 1.00/1.00.

Response history				
Step	Time	Action	State	Marks
1	8/03/19, 13:45	Started	Not complete	
2	11/03/19, 15:41	Submit: public class SpaceStation { public void dock() { System.out.println("Starman floats toward the space station and enters through the airlock"); } public void dock(String vehicle) { System.out.println("Starman enters the large airlock, piloting "+vehicle); } public void dock(String vehicle, String songName) { dock(vehicle); System.out.println("... and the station cranks up "+songName+" on the entertainment system"); } }	Correct	1.00
3	12/03/19, 12:46	Attempt finished submitting:	Correct	1.00

Information

If Statements

Sooner or later, we need to start making decisions in our code, and that is where if and switch statements come in.

If statements are used to determine what blocks of code should be executed if the condition statement evaluates to true or false.

If statements look like this in Java:

```
if (expression) {  
    // Evaluates if expression is true  
} else {  
    // Evaluates if expression is false  
}
```

If statements can be nested like this:

```
if (expression1) {  
    // Evaluates if expression1 is true  
    if (expression2) {  
        // Evaluates if expression1 and expression2 are true  
    } else {  
        // Evaluates if expression1 is true and expression2 is false  
    }  
}
```

So what is an expression? It is any kind of boolean logic. It may be as simple as a boolean typed variable holding true or false, or it could be multiple conditionals. Lets have a look.

```
boolean boolTrue = true;  
boolean boolFalse = false;  
int a = 10;  
  
// one variable examples  
if (boolTrue) // evaluates to true  
if (boolFalse) // evaluates to false  
  
// <, <=, >, >= all work as you expect  
if (a <= 10) // evaluates to true  
if (a < 10) // evaluates to false  
if (a > 10) // evaluates to false  
if (a >= 10) // evaluates to true  
  
// if you want multiple conditions, && is AND and || is OR.  
if (a <=10 && boolTrue) // evaluates to true, since BOTH are true  
if (a <= 10 || boolTrue) // evaluates to true, since one or more are true  
if (boolTrue && boolFalse) // evaluates to false  
if (boolTrue || boolFalse) // evaluates to true  
  
// you can negate with ! to invert boolean  
if (!boolFalse && boolTrue) // evaluates to true  
if (!(a <= 10) || boolFalse) // evaluates to false
```

Response history			
Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	11/03/19, 16:08	Seen	
3	12/03/19, 12:46	Attempt finished	

Question 3

Correct

Mark 1.00 out of 1.00

LiftoffWatch Class

Starman wants to know if the Falcon Heavy rocket is allowed to launch with the current weather conditions. Write a class called LiftoffWatch which implements the following features:

- a variable called "temperature" which is a double, and holds the current temperature.
- a variable called "weather", which is a String and holds the current weather state (Sunny, Cloudy or Rainy)
- A variable called "wind", which is a double and holds the wind speed.

There needs to be the following methods:

- void setTemp(double temp), which sets the value of temperature
- void setWeather(String state), which sets the weather
- void setWind(double speed), which sets the wind
- boolean canWeLaunch(), which determines if we can launch or not.

To be able to launch, we need:

- The temperature to be between 16.5 and 34.0 degrees (inclusive).
- If the weather is sunny, we can launch if the wind is not above 60 kilometers per hour.
- If the weather is cloudy we can launch if the wind is no more than 45 kilometers per hour.
- If the weather is rainy, we cannot launch.

For example:

Test	Result
LiftoffWatch launch = new LiftoffWatch(); launch.setTemp(27.0); launch.setWeather("Sunny"); launch.setWind(53); System.out.println(launch.canWeLaunch());	true

Answer: (penalty regime: 10, 20, ... %)

```
1 public class LiftoffWatch {
2     private double temperature=0;
3     private String weather="";
4     private double wind=0;
5
6     public void setTemp(double temp) {
7         temperature=temp;
8     }
9     public void setWeather(String state) {
10        weather = state;
11    }
12    public void setWind(double speed) {
13        wind=speed;
14    }
15    public boolean canWeLaunch() {
16        boolean canLaunch = true;
17        if (temperature > 34.0 || temperature < 16.5)
18            canLaunch = false;
19
20        if (canLaunch) {
21            if (weather == "Sunny" && wind <= 60.0)
22                canLaunch = true; //do nothing
23        }
24    }
25 }
```

	Test	Expected	Got	
✓	LiftoffWatch launch = new LiftoffWatch(); launch.setTemp(27.0); launch.setWeather("Sunny"); launch.setWind(53); System.out.println(launch.canWeLaunch());	true	true	✓
✓	LiftoffWatch launch = new LiftoffWatch(); launch.setTemp(27.0); launch.setWeather("Rainy"); launch.setWind(53); System.out.println(launch.canWeLaunch());	false	false	✓

	Test	Expected	Got	
✓	<pre>LiftoffWatch launch = new LiftoffWatch(); launch.setTemp(27.0); launch.setWeather("Cloudy"); launch.setWind(80); System.out.println(launch.canWeLaunch());</pre>	false	false	✓
✓	<pre>LiftoffWatch launch = new LiftoffWatch(); launch.setTemp(27.0); launch.setWeather("Cloudy"); launch.setWind(35); System.out.println(launch.canWeLaunch());</pre>	true	true	✓

Passed all tests! ✓

Question author's solution:

```
public class LiftoffWatch {

    public double temperature;
    public String weather;
    public double wind;

    public void setTemp(double temp) {
        temperature = temp;
    }

    public void setWeather(String state) {
        if (state == "Sunny" || state == "Cloudy" || state == "Rainy") {
            weather = state;
        }
    }

    public void setWind(double speed) {
        wind = speed;
    }

    public boolean canWeLaunch() {
        if (temperature >= 16.5 && temperature <= 34.0) {
            if (weather == "Sunny" && wind <= 60) {
                return true;
            } else if (weather == "Cloudy" && wind <= 45) {
                return true;
            } else if (weather == "Rainy") {
                return false;
            } else {
                return false;
            }
        } else {
            return false;
        }
    }

    public static void main(String[] args) {
        LiftoffWatch launch = new LiftoffWatch();

        launch.setTemp(27.0);
        launch.setWeather("Sunny");
        launch.setWind(53);
        System.out.println(launch.canWeLaunch());
    }
}
```

Correct

Marks for this submission: 1.00/1.00.

Response history				
Step	Time	Action	State	Marks
1	8/03/19, 13:45	Started	Not complete	
2	11/03/19, 16:08	Submit: public class LiftoffWatch { private double temperature=0; private String weather=""; private double wind=0; public void setTemp(double temp) { temperature=temp; } public void setWeather(String state) { weather = state; } public void setWind(double speed) { wind=speed; } public boolean canWeLaunch() { boolean canLaunch = true; if (temperature > 34.0 temperature < 16.5) canLaunch = false; if (canLaunch) { if (weather == "Sunny" && wind <= 60.0) canLaunch = true;//do nothing else if (weather == "Cloudy" && wind <= 45.0) canLaunch = true;//do nothing else canLaunch = false; } return canLaunch; } }	Correct	1.00
3	12/03/19, 12:46	Attempt finished submitting:	Correct	1.00

Switch Statements

Frequently you might come across a set of well defined conditions or have a state machine with well defined states. If you used an "if" statement for all of those states, the code might get a little messy and large, so this is where a "switch" statement comes in.

Lets say we have the following states: GROUNDED, FUELLING, LAUNCHING, ORBIT, STARBOUND.

We can collect these states in whats called an **enum** (we have not covered enum in the lectures). Enums are a type of their own and are used to give states a type. These are quite commonly used with switch statements.

An example, with a switch statement to make a decision based on what state we are in.

```
enum RocketState {
    GROUNDED, FUELLING, LAUNCHING, ORBIT, STARBOUND
}

public class Rocket {

    public void printRocketState(RocketState state) {
        switch (state) {
            case GROUNDED:
                System.out.println("Stored inside the hangar");
                break;
            case FUELLING:
                System.out.println("On the launch pad, fuelling.");
                break;
            case LAUNCHING:
                System.out.println("Launching into the sky");
                break;
            case ORBIT:
                System.out.println("Floating around the planet");
                break;
            case STARBOUND:
                System.out.println("Deep space here we come");
                break;
            default:
                System.out.println("Unknown state");
                break;
        }
    }

    public static void main(String[] args) {
        Rocket rocket = new Rocket();
        RocketState state = RocketState.FUELLING;
        rocket.printRocketState(state);
    }
}
```

We "switch" on a variable, and the block of code that the variable equates to is executed. Note that switch statements "fall-through". This is where one block of code is executed, and will go to the next one down. To stop this, place a break statement in to break out of the switch statement.

You don't have to use enum arrays with switch statements (see the examples in the lecture notes). It is just a way to make things a bit cleaner. A switch will also work with the byte, short, char, and int primitive data types. It also works with the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer.

Response history			
Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	11/03/19, 16:08	Seen	
3	12/03/19, 12:46	Attempt finished	

Question **4**

Correct

Mark 0.80 out of 1.00

Starman's Holiday (Using Enums)

Starman is on holiday, and decided to go to Mars for some time off. The tourist attractions on Mars are climbing MOUNTAINS, exploring VALLEYS, crossing ICE_CAPS, saying hello to CURIOSITY, and drinking some lemonade in the shade of a RED_SAND beach.

Starman needs some help keeping track of all these activities and what happens at those sites. Write an enum called MarsActivities to keep track of the activities, and a class called MarsHoliday that implements:

- void activityChooser(MarsActivities activity), which is a switch statement that implements:
 - MOUNTAINS - uses System.out.println to print the String "The air is no different up here than on the ground"
 - VALLEYS - uses System.out.println to print the String "Did I just see a Martian?"
 - ICE_CAPS - uses System.out.println to print the String "Maybe I can drink this"
 - CURIOSITY - uses System.out.println to print the String "Lets pat Curiosity", and then calls a method patCuriosity()
 - RED_SAND - uses System.out.println to print the String "That's hot!"
- void patCuriosity() - uses System.out.println to print the String "Pat pat pat" followed by a newline, and then "Curiosity seems happy"

Remember to put break; statements in, or you will be in for a fun time debugging!

For example:

Test	Result
MarsHoliday holiday = new MarsHoliday(); holiday.activityChooser(MarsActivities.MOUNTAINS); holiday.activityChooser(MarsActivities.VALLEYS); holiday.activityChooser(MarsActivities.ICE_CAPS); holiday.activityChooser(MarsActivities.CURIOSITY); holiday.activityChooser(MarsActivities.RED_SAND);	The air is no different up here than on the ground Did I just see a Martian? Maybe I can drink this Lets pat Curiosity Pat pat pat Curiosity seems happy That's hot!

Answer: (penalty regime: 10, 20, ... %)

```
1 enum MarsActivities {  
2     MOUNTAINS, VALLEYS, ICE_CAPS, CURIOSITY, RED_SAND;  
3 }  
4  
5 public class MarsHoliday {  
6     public void activityChooser(MarsActivities activity) {  
7         switch (activity) {  
8             case MOUNTAINS:  
9                 System.out.println("The air is no different up here than on the ground");  
10                break;  
11            case VALLEYS:  
12                System.out.println("Did I just see a Martian?");  
13                break;  
14            case ICE_CAPS:  
15                System.out.println("Maybe I can drink this");  
16                break;  
17            case CURIOSITY:  
18                System.out.println("Lets pat Curiosity");  
19                patCuriosity();  
20                break;  
21            case RED_SAND:  
22                System.out.println("That's hot!");  
23        }  
24    }  
25}
```

	Test	Expected	Got	
✓	MarsHoliday holiday = new MarsHoliday(); holiday.activityChooser(MarsActivities.MOUNTAINS); holiday.activityChooser(MarsActivities.VALLEYS); holiday.activityChooser(MarsActivities.ICE_CAPS); holiday.activityChooser(MarsActivities.CURIOSITY); holiday.activityChooser(MarsActivities.RED_SAND);	The air is no different up here than on the ground Did I just see a Martian? Maybe I can drink this Lets pat Curiosity Pat pat pat Curiosity seems happy That's hot!	The air is no different up here than on the ground Did I just see a Martian? Maybe I can drink this Lets pat Curiosity Pat pat pat Curiosity seems happy That's hot!	✓

	Test	Expected	Got	
✔	MarsHoliday holiday = new MarsHoliday(); holiday.activityChooser(MarsActivities.CURIOSITY); holiday.activityChooser(MarsActivities.MOUNTAINS); holiday.activityChooser(MarsActivities.ICE_CAPS); holiday.activityChooser(MarsActivities.VALLEYS); holiday.activityChooser(MarsActivities.RED_SAND);	Lets pat Curiosity Pat pat pat Curiosity seems happy The air is no different up here than on the ground Maybe I can drink this Did I just see a Martian? That's hot!	Lets pat Curiosity Pat pat pat Curiosity seems happy The air is no different up here than on the ground Maybe I can drink this Did I just see a Martian? That's hot!	✔

Passed all tests! ✔

Question author's solution:

```
enum MarsActivities {  
    MOUNTAINS, VALLEYS, ICE_CAPS, CURIOSITY, RED_SAND  
}  
  
public class MarsHoliday {  
  
    public void activityChooser(MarsActivities activity) {  
        switch (activity) {  
            case MOUNTAINS:  
                System.out.println("The air is no different up here than on the ground");  
                break;  
            case VALLEYS:  
                System.out.println("Did I just see a Martian?");  
                break;  
            case ICE_CAPS:  
                System.out.println("Maybe I can drink this");  
                break;  
            case CURIOSITY:  
                System.out.println("Lets pat Curiosity");  
                patCuriosity();  
                break;  
            case RED_SAND:  
                System.out.println("That's hot!");  
                break;  
            default:  
                System.out.println("ERROR");  
                break;  
        }  
    }  
  
    public void patCuriosity() {  
        System.out.println("Pat pat pat");  
        System.out.println("Curiosity seems happy");  
    }  
  
    public static void main(String[] args) {  
        MarsHoliday holiday = new MarsHoliday();  
        holiday.activityChooser(MarsActivities.MOUNTAINS);  
        holiday.activityChooser(MarsActivities.VALLEYS);  
        holiday.activityChooser(MarsActivities.ICE_CAPS);  
        holiday.activityChooser(MarsActivities.CURIOSITY);  
        holiday.activityChooser(MarsActivities.RED_SAND);  
    }  
}
```

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.80/1.00**.

Response history				
Step	Time	Action	State	Marks
1	8/03/19, 13:45	Started	Not complete	
2	11/03/19, 16:22	Submit: public enum MarsActivities { MOUNTAINS, VALLEYS, ICE_CAPS, CURIOSITY, RED_SAND; } public class MarsHoliday { public void activityChooser(MarsActivities activity) { switch (activity) { case MOUNTAINS: System.out.println("The air is no different up here than on the ground"); break; case VALLEYS: System.out.println("Did I just see a Martian?"); break; case ICE_CAPS: System.out.println("Maybe I can drink this"); break; case CURIOSITY: System.out.println("Lets pat Curiosity"); patCuriosity(); break; case RED_SAND: System.out.println("That's hot!"); break; default: break; } } public void patCuriosity() { System.out.println("Pat pat pat\nCuriosity seems happy"); } }	Incorrect	0.00
3	11/03/19, 16:22	Submit: public class MarsHoliday { public void activityChooser(MarsActivities activity) { switch (activity) { case MOUNTAINS: System.out.println("The air is no different up here than on the ground"); break; case VALLEYS: System.out.println("Did I just see a Martian?"); break; case ICE_CAPS: System.out.println("Maybe I can drink this"); break; case CURIOSITY: System.out.println("Lets pat Curiosity"); patCuriosity(); break; case RED_SAND: System.out.println("That's hot!"); break; default: break; } } public void patCuriosity() { System.out.println("Pat pat pat\nCuriosity seems happy"); } }	Incorrect	0.00
4	11/03/19, 16:23	Submit: enum MarsActivities { MOUNTAINS, VALLEYS, ICE_CAPS, CURIOSITY, RED_SAND; } public class MarsHoliday { public void activityChooser(MarsActivities activity) { switch (activity) { case MOUNTAINS: System.out.println("The air is no different up here than on the ground"); break; case VALLEYS: System.out.println("Did I just see a Martian?"); break; case ICE_CAPS: System.out.println("Maybe I can drink this"); break; case CURIOSITY: System.out.println("Lets pat Curiosity"); patCuriosity(); break; case RED_SAND: System.out.println("That's hot!"); break; default: break; } } public void patCuriosity() { System.out.println("Pat pat pat\nCuriosity seems happy"); } }	Correct	0.80
5	12/03/19, 12:46	Attempt finished submitting:	Correct	0.80

Information

For loops

Let's take a look at some loops, starting with the for-loop. for-loops in Java typically always involve a counter, an expression and an increment. Lets have a look at the syntax.

```
for(counter; expression; increment) {  
    // statements  
}
```

Counters are typically integers. Expressions can be any expressions that we talked about in the if-statement section. The increment usually modifies the counter so we will eventually break out of the loop.

A physical example:

```
for (int i = 0; i < 100; i++) {  
    System.out.println("Hello there");  
}
```

This loop prints "Hello there" one hundred times standard output.

Now, technically you do not have to specify the counter or expression or increment. Please avoid doing this as it can be bad style and confusing. If you need to loop and want to omit fields, use a while-loop instead. In this course, and in your work in industry, if you use a for-loop, it is always best practice to specify the counter, expression and increment sections.

Response history

Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	12/03/19, 00:51	Seen	
3	12/03/19, 12:46	Attempt finished	

Question **5**

Correct

Mark 0.60 out of 1.00

LocationBeacon class

Starman has been thinking about installing a location beacon in his roadster so scientists from Earth can keep track of his position in the Universe. All the beacon does is emit a "heartbeat" signal with a current timestamp and distance.

Today, the time is 14:31 hours (24 hour time) and Starman is 300,000 kilometres away from Earth (relatively close!) He travels 60,000km away from the earth every hour in his Red Tesla Roadster. Output a heartbeat every ten minutes of the time and his distance away from Earth.

Write a class called LocationBeacon which implements:

- a variable called hours of type int, to keep track of hours.
- a variable called minutes of type int to keep track of minutes.
- a variable called distance of type int to keep track of distance (in km).
- void heartBeat(int startHour, int startMinute, int startDistance, int speed, int duration) which outputs "[<hour>: <minute>] Starman is <distance>km away from Earth".
start distance - how far Starman currently is from Earth, in km.
duration - is how many minutes the heartbeat should be output for.
speed - how fast Starman is moving in km/h.
The heartbeat should not be output for time zero.

Note, you will need if-statements to keep track of hours flowing over 24 and minutes flowing over 60.

For example:

Test	Result
LocationBeacon beacon = new LocationBeacon(); beacon.heartBeat(14, 31, 300000, 60000, 60);	[14:41] Starman is 310000km away from Earth [14:51] Starman is 320000km away from Earth [15:1] Starman is 330000km away from Earth [15:11] Starman is 340000km away from Earth [15:21] Starman is 350000km away from Earth [15:31] Starman is 360000km away from Earth

Answer: (penalty regime: 10, 20, ... %)

```
1 public class LocationBeacon {
2     private int hours=0;
3     private int minutes=0;
4     private int distance=0;
5
6     public void heartBeat(int startHour, int startMinute, int startDistance, int speed, int duration) {
7         hours = startHour;
8         minutes = startMinute;
9         distance = startDistance;
10
11         for (int i = 0; i < duration/10; i++) {
12             //System.out.println(minutes);
13
14             if (minutes < 50) {
15                 minutes += 10;
16
17             }
18             else if (minutes >= 50) {
19                 minutes -= 50;
20                 if (hours < 23)
21                     hours +=1 ;
22                 minutes = 10;
23             }
24             distance += speed;
25             System.out.println(hours+":"+minutes+" Starman is "+distance+"km away from Earth");
26         }
27     }
28 }
```

	Test	Expected	Got	
✓	LocationBeacon beacon = new LocationBeacon();\t beacon.heartBeat(14, 31, 300000, 60000, 60);	[14:41] Starman is 310000km away from Earth [14:51] Starman is 320000km away from Earth [15:1] Starman is 330000km away from Earth [15:11] Starman is 340000km away from Earth [15:21] Starman is 350000km away from Earth [15:31] Starman is 360000km away from Earth	[14:41] Starman is 310000km away from Earth [14:51] Starman is 320000km away from Earth [15:1] Starman is 330000km away from Earth [15:11] Starman is 340000km away from Earth [15:21] Starman is 350000km away from Earth [15:31] Starman is 360000km away from Earth	✓

	Test	Expected	Got	
✓	<pre>LocationBeacon beacon = new LocationBeacon();\t beacon.heartBeat(23, 28, 630000, 10000, 90);</pre>	<pre>[23:38] Starman is 631666km away from Earth [23:48] Starman is 633332km away from Earth [23:58] Starman is 634998km away from Earth [0:8] Starman is 636664km away from Earth [0:18] Starman is 638330km away from Earth [0:28] Starman is 639996km away from Earth [0:38] Starman is 641662km away from Earth [0:48] Starman is 643328km away from Earth [0:58] Starman is 644994km away from Earth</pre>	<pre>[23:38] Starman is 631666km away from Earth [23:48] Starman is 633332km away from Earth [23:58] Starman is 634998km away from Earth [0:8] Starman is 636664km away from Earth [0:18] Starman is 638330km away from Earth [0:28] Starman is 639996km away from Earth [0:38] Starman is 641662km away from Earth [0:48] Starman is 643328km away from Earth [0:58] Starman is 644994km away from Earth</pre>	✓
✓	<pre>LocationBeacon beacon = new LocationBeacon();\t beacon.heartBeat(9, 17, 1000000, 20090, 20);</pre>	<pre>[9:27] Starman is 1003348km away from Earth [9:37] Starman is 1006696km away from Earth</pre>	<pre>[9:27] Starman is 1003348km away from Earth [9:37] Starman is 1006696km away from Earth</pre>	✓

Passed all tests! ✓

Question author's solution:

```
public class LocationBeacon {
    public int hours;
    public int minutes;
    public int distance;

    public void heartBeat(int startHour, int startMinute, int startDistance, int speed, int duration) {
        hours = startHour;
        minutes = startMinute;
        distance = startDistance;

        for (int i = 0; i < duration / 10; i++) {
            distance += speed / 6;
            minutes += 10;

            if (minutes >= 60) {
                minutes = minutes % 60;
                hours += 1;
            }
            if (hours >= 24) {
                hours = hours % 24;
            }

            System.out.println "[" + hours + ":" + minutes + " ] " +
                "Starman is " + distance + "km away from Earth";
        }
    }

    public static void main(String[] args) {
        LocationBeacon beacon = new LocationBeacon();

        beacon.heartBeat(14, 31, 300000, 60000, 60);
        beacon.heartBeat(23, 28, 630000, 10000, 90);
        beacon.heartBeat(9, 17, 1000000, 20090, 20);
    }
}
```

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.60/1.00**.

Response history				
Step	Time	Action	State	Marks
1	8/03/19, 13:45	Started	Not complete	
2	12/03/19, 00:51	Submit: public class LocationBeacon { private int hours=0; private int minutes=0; private int distance=0;//in km public void heartBeat(int startHour, int startMinute, int startDistance, int speed, int duration) { hours = startHour; minutes = startMinute; distance = startDistance; for (int i = 0; i < duration/10; i++) { if (minutes < 60 && minutes >= 0) { minutes += 10; } else if (minutes >= 60) { minutes -= 60; hours +=1 ; } distance += speed/6; String string = String.format("[%d:%d] Starman is %dkm away from Earth", hours, minutes, distance); System.out.println(string); //System.out.println("["+hours+": "+minutes+"] Starman is "+distance+"km away from Earth"); } } }	Incorrect	0.00
3	12/03/19, 01:06	Submit: package Lab3; public class LocationBeacon { private int hours=0; private int minutes=0; private int distance=0; public void heartBeat(int startHour, int startMinute, int startDistance, int speed, int duration) { hours = startHour; minutes = startMinute; distance = startDistance; for (int i = 0; i < duration/10; i++) { System.out.println(minutes); if (minutes < 50) { minutes += 10; } else if (minutes >= 50) { minutes -= 60; hours +=1 ; continue; } distance += speed/6; //String string = String.format("[%d:%d] Starman is %dkm away from Earth", hours, minutes, distance); //System.out.println(string); System.out.println("["+hours+": "+minutes+"] Starman is "+distance+"km away from Earth"); } } }	Incorrect	0.00
4	12/03/19, 01:06	Submit: public class LocationBeacon { private int hours=0; private int minutes=0; private int distance=0; public void heartBeat(int startHour, int startMinute, int startDistance, int speed, int duration) { hours = startHour; minutes = startMinute; distance = startDistance; for (int i = 0; i < duration/10; i++) { System.out.println(minutes); if (minutes < 50) { minutes += 10; } else if (minutes >= 50) { minutes -= 60; hours +=1 ; continue; } distance += speed/6; //String string = String.format("[%d:%d] Starman is %dkm away from Earth", hours, minutes, distance); //System.out.println(string); System.out.println("["+hours+": "+minutes+"] Starman is "+distance+"km away from Earth"); } } }	Incorrect	0.00
5	12/03/19, 01:07	Submit: public class LocationBeacon { private int hours=0; private int minutes=0; private int distance=0; public void heartBeat(int startHour, int startMinute, int startDistance, int speed, int duration) { hours = startHour; minutes = startMinute; distance = startDistance; for (int i = 0; i < duration/10; i++) { if (minutes < 50) { minutes += 10; } else if (minutes >= 50) { minutes -= 60; hours +=1 ; continue; } distance += speed/6; //String string = String.format("[%d:%d] Starman is %dkm away from Earth", hours, minutes, distance); //System.out.println(string); System.out.println("["+hours+": "+minutes+"] Starman is "+distance+"km away from Earth"); } } }	Incorrect	0.00
6	12/03/19, 01:12	Submit: public class LocationBeacon { private int hours=0; private int minutes=0; private int distance=0; public void heartBeat(int startHour, int startMinute, int startDistance, int speed, int duration) { hours = startHour; minutes = startMinute; distance = startDistance; for (int i = 0; i < duration/10; i++) { //System.out.println(minutes); if (minutes < 50) { minutes += 10; } else if (minutes >= 50) { minutes -= 50; if (hours < 23) hours +=1 ; else hours -= 23; } distance += speed/6; //String string = String.format("[%d:%d] Starman is %dkm away from Earth", hours, minutes, distance); //System.out.println(string); System.out.println("["+hours+": "+minutes+"] Starman is "+distance+"km away from Earth"); } } }	Correct	0.60
7	12/03/19, 12:46	Attempt finished submitting:	Correct	0.60

Information

While loops

Lets take a look at while loops. A while loop iterates as long as the expression is true. The loop is only broken when the expression turns false.

Syntax:

```
while (expression) {  
    // statements  
}
```

The expression can be single variable booleans, or any expression type we saw in the if-statement section. A physical example:

```
while (launchNotReady && countdownTimer > 1800) {  
    readyRocket();  
}
```

Response history

Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	12/03/19, 00:51	Seen	
3	12/03/19, 12:46	Attempt finished	

Question **6**

Correct

Mark 0.70 out of 1.00

StarmanSummits (while loops)

Starman is enjoying his holiday, and is currently trying to summit the largest mountain on Mars, Olympus Mons. Starman is having a tough time, being a humanoid robot in an awkward space suit, trying to climb a mountain twice the size of Mount Everest. The mountain is 21,287.4 meters high. Starman can run up the mountain in 16.4 meter dashes before he has to stop to take a rest. When he stops, he slides down the mountain 4.3 meters since he is on a gravel face. How many dashes does it take Starman to run up the mountain?

Write a class called StarmanSummits which implements the following method:

- void climbMountain(double height, double dash, double slide), where height is the total height of the mountain, dash is the length Starman can run for before stopping, and slide is how far starman slides down the mountain at each stop.

The method should print "Starman needs to dash <attempts> times before he reaches the top of the mountain"

Note your code will be tested with other heights, dashes and slides. (So don't hardcode)

For example:

Test	Result
StarmanSummits starman = new StarmanSummits(); starman.climbMountain(21287.4, 16.4, 4.3);	Starman needs to dash 1759 times before he reaches the top of the mountain

Answer: (penalty regime: 10, 20, ... %)

```
1 public class StarmanSummits {
2     public void climbMountain(double height, double dash, double slide) {
3         double currHeight = 0;
4         int attempt=0;
5         while (currHeight < height) {
6             currHeight+=dash;
7             attempt++;
8             if (currHeight < height)
9                 currHeight-=slide;
10        }
11
12        System.out.println("Starman needs to dash "+attempt+" times before he reaches the t
13    }
14 }
```

	Test	Expected	Got	
✓	StarmanSummits starman = new StarmanSummits(); starman.climbMountain(21287.4, 16.4, 4.3);	Starman needs to dash 1759 times before he reaches the top of the mountain	Starman needs to dash 1759 times before he reaches the top of the mountain	✓
✓	StarmanSummits starman = new StarmanSummits(); starman.climbMountain(123456, 54.7, 2.8);	Starman needs to dash 2379 times before he reaches the top of the mountain	Starman needs to dash 2379 times before he reaches the top of the mountain	✓
✓	StarmanSummits starman = new StarmanSummits(); starman.climbMountain(10, 2, 1);	Starman needs to dash 9 times before he reaches the top of the mountain	Starman needs to dash 9 times before he reaches the top of the mountain	✓

Passed all tests! ✓

Question author's solution:


```
public class StarmanSummits {  
  
    void climbMountain(double height, double dash, double slide) {  
        double attained = 0.0;  
        int attempts = 0;  
  
        while (attained < height) {  
            attempts += 1;  
            attained += dash;  
            if (attained < height) {  
                // if attained >= height, then we are on top of mountain  
                attained -= slide;  
            }  
        }  
  
        System.out.println("Starman needs to dash " + attempts +  
            " times before he reaches the top of the mountain");  
    }  
  
    public static void main(String[] args) {  
        StarmanSummits starman = new StarmanSummits();  
        starman.climbMountain(21287.4, 16.4, 4.3);  
        starman.climbMountain(123456, 54.7, 2.8);  
        starman.climbMountain(10, 2, 1);  
    }  
}
```

Correct
Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.70/1.00**.

Response history				
Step	Time	Action	State	Marks
1	8/03/19, 13:45	Started	Not complete	
2	12/03/19, 01:23	Submit: public class StarmanSummits { public void climbMountain(double height, double dash, double slide) { double currHeight = 0; int attempt=-1; while (currHeight < height) { currHeight+=dash; currHeight-=slide; attempt++; } System.out.println("Starman needs to dash "+attempt+" times before he reaches the top of the mountain"); } }	Incorrect	0.00
3	12/03/19, 01:32	Submit: public class StarmanSummits { public void climbMountain(double height, double dash, double slide) { double currHeight = 0; int attempt=-1; while (currHeight < height) { currHeight+=dash; attempt++; if (currHeight != height) currHeight-=slide; } System.out.println("Starman needs to dash "+attempt+" times before he reaches the top of the mountain"); } }	Incorrect	0.00
4	12/03/19, 01:32	Submit: public class StarmanSummits { public void climbMountain(double height, double dash, double slide) { double currHeight = 0; int attempt=0; while (currHeight < height) { currHeight+=dash; attempt++; if (currHeight != height) currHeight-=slide; } System.out.println("Starman needs to dash "+attempt+" times before he reaches the top of the mountain"); } }	Incorrect	0.00
5	12/03/19, 01:33	Submit: public class StarmanSummits { public void climbMountain(double height, double dash, double slide) { double currHeight = 0; int attempt=0; while (currHeight < height) { currHeight+=dash; attempt++; if (currHeight < height) currHeight-=slide; } System.out.println("Starman needs to dash "+attempt+" times before he reaches the top of the mountain"); } }	Correct	0.70
6	12/03/19, 12:46	Attempt finished submitting:	Correct	0.70

Information

Getting into multiple objects

Being object-orientated makes Java a very powerful programming language, as classes and instances offer you the opportunity to "model the real world". As we have seen before, in Java you can have more than one instance of a class. This lets you have objects of the same type, that do similar things, but might hold different data in class attributes.

Response history			
Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	12/03/19, 12:42	Seen	
3	12/03/19, 12:46	Attempt finished	

Multiple objects example and the toString() method

The example we will look at is an Astronaut. Astronauts are people, they have names, ages, favourite food and space mission information. We can easily have multiple astronauts, since in the real world we can just add more people, give them a space suit and call them astronauts. Object-orientated languages let us model the real world, so it is easy to add astronauts too.

```
public class Astronaut {

    public String name;
    public int age;
    public String favouriteFood;
    public String spaceMission;

    public String toString() {
        return "Hi, my name is " + name + " and I am " + age + " years old. " +
            "My favourite food is " + favouriteFood + " and I am on mission "
            + spaceMission + ".";
    }

    public static void main(String[] args) {
        Astronaut spaceboy = new Astronaut();
        spaceboy.name = "Spaceboy";
        spaceboy.age = 20;
        spaceboy.favouriteFood = "Hash browns";
        spaceboy.spaceMission = "0001 to Mars";

        Astronaut spacegirl = new Astronaut();
        spacegirl.name = "Spacegirl";
        spacegirl.age = 21;
        spacegirl.favouriteFood = "Pizza";
        spacegirl.spaceMission = "0002 to the Moon";

        System.out.println(spaceboy);
        System.out.println(spacegirl);
    }
}
```

This outputs:

```
Hi, my name is Spaceboy and I am 20 years old. My favourite food is Hash browns and I am on mission 0001 to Mars.
Hi, my name is Spacegirl and I am 21 years old. My favourite food is Pizza and I am on mission 0002 to the Moon.
```

Be making a new variable, and instantiating a new Astronaut() to place inside that variable, we are able to have multiple objects of the same type. Each object has access to the same methods of the class, but their attributes are separate from each other, meaning that updates are on a per object basis.

The Magic of toString()

Another thing to note is how we passed an object directly into the print statement, and we got a nicely formatted string representation back. What is this magic you say? It is the magic of toString().

Every class has a toString() method. When Java tries to turn an object into a string, it automatically calls the toString method. If the class does not have one, then we get the basic boring toString in java.lang.Object. This prints the class name and the instance number. We get the nice output like above by implementing our own toString() method.

toString() is super useful. Expect to use it pretty much everywhere in the code you write.

Response history			
Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	12/03/19, 12:42	Seen	
3	12/03/19, 12:46	Attempt finished	

Question **7**

Correct

Mark 1.00 out of 1.00

GreenAlien class

Mars attacks! No really, some little green Aliens have descended from the heavens and are starting to be a nuisance to Starman, as the Aliens are trying to get his attention by eating packets of loud pop rocks infront of him. Starman has decided to make a class which describes these little green visitors so he can keep track of them.

Write a class called GreenAlien which has the following attributes:

- an attribute called name of type String. This holds the Aliens name.
- an attribute called eyeCount of type int. This holds the number of eyes an Alien has.
- an attribute called favouriteCandy of type String. This holds the Aliens perceived favourite candy.

And Methods:

- String toString(), which returns a String of "This Alien is called <name> and has <eyeCount> eyes. Gross. It seems to enjoy eating <favouriteCandy>".

When you test your Class, make sure to have multiple instances of GreenAliens in your main() method. It's good to get used to having multiple objects of your code.

For example:

Test	Result
<pre>GreenAlien gwerp = new GreenAlien(); gwerp.name = "Gwerp"; gwerp.eyeCount = 4; gwerp.favouriteCandy = "Marshmallows"; System.out.println(gwerp);</pre>	<pre>This Alien is called Gwerp and has 4 eyes. Gross. It seems to enjoy eating Marshmallows</pre>

Answer: (penalty regime: 10, 20, ... %)

```
1 public class GreenAlien {
2     public String name="";
3     public int eyeCount=0;
4     public String favouriteCandy="";
5
6     public String toString() {
7         return "This Alien is called "+name+" and has "+eyeCount+" eyes. Gross. It seems to
8     }
9 }
10
```

	Test	Expected	Got	
✓	<pre>GreenAlien gwerp = new GreenAlien(); gwerp.name = "Gwerp"; gwerp.eyeCount = 4; gwerp.favouriteCandy = "Marshmallows"; System.out.println(gwerp);</pre>	<pre>This Alien is called Gwerp and has 4 eyes. Gross. It seems to enjoy eating Marshmallows</pre>	<pre>This Alien is called Gwerp and has 4 eyes. Gross. It seems to enjoy eating Marshmallows</pre>	✓

	Test	Expected	Got	
✓	<div>GreenAlien blarg = new GreenAlien(); blarg.name = "Blarg"; blarg.eyeCount = 3; blarg.favouriteCandy = "Pop Rocks";</div> <div>GreenAlien hloff = new GreenAlien(); hloff.name = "Hloff"; hloff.eyeCount = 5; hloff.favouriteCandy = "Pineapple Lumps";</div> <div>System.out.println(blarg); System.out.println(hloff);</div>	<div>This Alien is called Blarg and has 3 eyes. Gross. It seems to enjoy eating Pop Rocks</div> <div>This Alien is called Hloff and has 5 eyes. Gross. It seems to enjoy eating Pineapple Lumps</div>	<div>This Alien is called Blarg and has 3 eyes. Gross. It seems to enjoy eating Pop Rocks</div> <div>This Alien is called Hloff and has 5 eyes. Gross. It seems to enjoy eating Pineapple Lumps</div>	✓

Passed all tests! ✓

Question author's solution:

```
public class GreenAlien {  
  
    public String name;  
    public int eyeCount;  
    public String favouriteCandy;  
  
    @Override  
    public String toString() {  
        return String.format("This Alien is called %s and has %d eyes. Gross. It seems to enjoy eating %s",  
name, eyeCount, favouriteCandy);  
    }  
}
```

Correct
Marks for this submission: 1.00/1.00.

Response history				
Step	Time	Action	State	Marks
1	8/03/19, 13:45	Started	Not complete	
2	12/03/19, 12:42	Submit: public class GreenAlien { public String name=""; public int eyeCount=0; public String favouriteCandy=""; public String toString() { return "This Alien is called "+name+" and has "+eyeCount+" eyes. Gross. It seems to enjoy eating "+favouriteCandy; } }	Correct	1.00
3	12/03/19, 12:46	Attempt finished submitting:	Correct	1.00

More details about constructors

Okay, so that was a very short and basic class, both `Astronaut` and `GreenAlien`. We will spend the next few questions making nice maintainable object oriented Java code.

You may have noticed that making variables / instances of a class, and then setting all the necessary variables with direct assignment is a little tedious. Lets learn about a better way, called Constructors.

Here's how we instantiate objects:

```
Astronaut starman = new Astronaut();
```

The first `Astronaut` indicates the objects type. The `starman` represents the objects name. At the moment, you can think of it as an empty container that can fit objects of type `Astronaut`, but it doesn't come with one by default. This is why we have to instantiate / create a new object, and then place it in the variable, or inside the container. So we do this with *new*. What *new* does is call the method next to it, and returns a brand new object, and places it inside the variable.

But how is `Astronaut()` a method? Wasn't it a class? Well. `Astronaut` is the class. `Astronaut()` is the *constructor*.

A constructor is a method that has the same name as the class. This method is always called during the objects creation. You typically use constructors to initialise a object, before you start doing other tasks. Lets have a look to see how they are used.

```
public class Astronaut {

    public String name;
    public int age;
    public String favouriteFood;
    public String spaceMission;

    public Astronaut() {
        name = "Starman";
        age = 23;
        favouriteFood = "Burgers";
        spaceMission = "0000 space exploration";
    }

    public Astronaut(String tempName, int tempAge, String tempFood, String tempMission) {
        name = tempName;
        age = tempAge;
        favouriteFood = tempFood;
        spaceMission = tempMission;
    }

    public String toString() {
        return "Hi, my name is " + name + " and I am " + age + " years old. " +
            "My favourite food is " + favouriteFood + " and I am on mission "
            + spaceMission + ".";
    }

    public static void main(String[] args) {
        Astronaut starman = new Astronaut();
        Astronaut spaceboy = new Astronaut("Spaceboy", 20, "Hash browns", "0001 to Mars");
        Astronaut spacegirl = new Astronaut("Spacegirl", 21, "Pizza", "0002 to the Moon");

        System.out.println(starman);
        System.out.println(spaceboy);
        System.out.println(spacegirl);
    }
}
```

Which outputs:

```
Hi, my name is Starman and I am 23 years old. My favourite food is Burgers and I am on mission 0000 space exploration.
Hi, my name is Spaceboy and I am 20 years old. My favourite food is Hash browns and I am on mission 0001 to Mars.
Hi, my name is Spacegirl and I am 21 years old. My favourite food is Pizza and I am on mission 0002 to the Moon.
```

So what's different?

We created our objects using constructors. The `starman` variable was created using the default constructor. As you can see, the default constructor is great at setting default values. When you need customisation, you can see that there is another `Astronaut()` method, but it is overloaded with different parameters. You can overload constructors as much or as little as you wish. In this case, we are using them to set the values that makes an `Astronaut` special.

Constructors may look clunky, but when your code isn't trivial like these lab questions, they help more than you think.

Response history			
Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	12/03/19, 12:42	Seen	
3	12/03/19, 12:46	Attempt finished	

Question **8**

Correct

Mark 1.00 out of 1.00

Expanding on GreenAlien

In Java we can multiple constructors for a class, allowing an object to be instantiated in different ways.

Add constructors to the GreenAlien class.

Add:

- GreenAlien() which defaults name to "Kloup", eyeCount to 6, and favouriteCandy to "Lollypops".
- GreenAlien(String tempName, int tempEye, String tempCandy), which sets the values for name, eyeCount and favouriteCandy.

Make sure to test with multiple instances!

Answer: (penalty regime: 10, 20, ... %)

```
1 public class GreenAlien {
2     public String name="";
3     public int eyeCount=0;
4     public String favouriteCandy="";
5
6     public GreenAlien() {
7         name="Kloup";eyeCount=6;favouriteCandy="Lollypops";
8     }
9     public GreenAlien(String tempName, int tempEye, String tempCandy) {
10         name=tempName;eyeCount=tempEye;favouriteCandy=tempCandy;
11     }
12
13
14     public String toString() {
15         return "This Alien is called "+name+" and has "+eyeCount+" eyes. Gross. It seems to
16     }
17 }
18
```

	Test	Expected	Got	
✓	<div>GreenAlien kloup = new GreenAlien();</div> <div>GreenAlien gwerp = new GreenAlien("Gwerp", 4, "Marshmallows");</div> <div>GreenAlien blarg = new GreenAlien("Blarg", 3, "Pop Rocks");</div> <div>GreenAlien hloff = new GreenAlien("Hloff", 5, "Pineapple Lumps");</div> <div>System.out.println(kloup);</div> <div>System.out.println(gwerp);</div> <div>System.out.println(blarg);</div> <div>System.out.println(hloff);</div>	<div>This Alien is called Kloup and has 6 eyes. Gross. It seems to enjoy eating Lollypops</div> <div>This Alien is called Gwerp and has 4 eyes. Gross. It seems to enjoy eating Marshmallows</div> <div>This Alien is called Blarg and has 3 eyes. Gross. It seems to enjoy eating Pop Rocks</div> <div>This Alien is called Hloff and has 5 eyes. Gross. It seems to enjoy eating Pineapple Lumps</div>	<div>This Alien is called Kloup and has 6 eyes. Gross. It seems to enjoy eating Lollypops</div> <div>This Alien is called Gwerp and has 4 eyes. Gross. It seems to enjoy eating Marshmallows</div> <div>This Alien is called Blarg and has 3 eyes. Gross. It seems to enjoy eating Pop Rocks</div> <div>This Alien is called Hloff and has 5 eyes. Gross. It seems to enjoy eating Pineapple Lumps</div>	✓

Passed all tests! ✓

Question author's solution:


```
public class GreenAlien {

    public String name;
    public int eyeCount;
    public String favouriteCandy;

    public GreenAlien() {
        name = "Kloup";
        eyeCount = 6;
        favouriteCandy = "Lollypops";
    }

    public GreenAlien(String tempName, int tempEye, String tempCandy) {
        name = tempName;
        eyeCount = tempEye;
        favouriteCandy = tempCandy;
    }

    @Override
    public String toString() {
        return String.format("This Alien is called %s and has %d eyes. Gross. It seems to enjoy eating %s",
name, eyeCount, favouriteCandy);
    }

}
```

Correct
Marks for this submission: 1.00/1.00.

Response history				
Step	Time	Action	State	Marks
1	8/03/19, 13:45	Started	Not complete	
2	12/03/19, 12:46	Submit: public class GreenAlien { public String name=""; public int eyeCount=0; public String favouriteCandy=""; public GreenAlien() { name="Kloup";eyeCount=6;favouriteCandy="Lollypops"; } public GreenAlien(String tempName, int tempEye, String tempCandy) { name=tempName;eyeCount=tempEye;favouriteCandy=tempCandy; } public String toString() { return "This Alien is called "+name+" and has "+eyeCount+" eyes. Gross. It seems to enjoy eating "+favouriteCandy; } }	Correct	1.00
3	12/03/19, 12:46	Attempt finished submitting:	Correct	1.00

Information

Bringing it all together...

Time to bring it all together. You can use other classes in your Java project, just by using them as standard variables in your code.

For example, we could make a RedTeslaRoadster class and make Starman a driver of the class.

```
public class RedTeslaRoadster {
    private Astronaut driver;
    private Astronaut passenger;

    public void setDriver(Astronaut newDriver) {
        driver = newDriver;
    }

    public void setPassenger(Astronaut newPassenger) {
        passenger = newPassenger;
    }

    public String toString() {
        return "The current occupants of the car are: \n" + driver + "\nand\n" + passenger;
    }

    public static void main(String[] args) {
        RedTeslaRoadster redCar = new RedTeslaRoadster();

        Astronaut starman = new Astronaut();
        Astronaut starDog = new Astronaut("Stardog", 4, "Bones", "0000 space exploration");

        redCar.setDriver(starman);
        redCar.setPassenger(starDog);

        System.out.println(redCar);
    }
}
```

Which prints

```
The current occupants of the car are:
Hi, my name is Starman and I am 23 years old. My favourite food is Burgers and I am on mission 0000 space
exploration.
and
Hi, my name is Stardog and I am 4 years old. My favourite food is Bones and I am on mission 0000 space
exploration.
```

You can make objects of any class that is located in the directory which the current class is located in. In order to use other classes, you would need to either import the specific class, or import the package that class is located in.

Response history			
Step	Time	Action	State
1	8/03/19, 13:45	Started	
2	12/03/19, 12:42	Seen	
3	12/03/19, 12:46	Attempt finished	

◀ JAR Example

Jump to...

Quiz 4 - Inheritance, Interfaces, and Collections (old) ▶