Computer Science 101 S1

Lecture 18

Contents

Coursebook: §14.2.1, §14.4.6, §15.9

Review

Complete TrafficLight class on slides 6 and 7.
Every TrafficLight instance contains two int values:

a number which is either 0, 1, 2
indicating whether the TrafficLight is
showing "GREEN", "ORANGE" or "RED".

the ID number of the TrafficLight.

Review

When a TrafficLight instance is created, the ID number is passed in as a parameter. The colour which the TrafficLight is initially showing is a random number of 0, 1, or 2.

The change() instance method should increase the colour value by 1. If the colour value is 2 then the colour value should change to 0.

Testing - TrafficLight class

Executing the following code with the completed TrafficLight class produces the output shown below:

```
TrafficLight t1 = new TrafficLight(334);
for( int i=0; i<5; i++) {
      t1.change();
      System.out.println(i + " " + t1.toString());
}
```

```
>java TrafficLight
0 Traffic light: 334, showing: GREEN
1 Traffic light: 334, showing: ORANGE
2 Traffic light: 334, showing: RED
3 Traffic light: 334, showing: GREEN
4 Traffic light: 334, showing: ORANGE
```


Ex01 - TrafficLight class

```
public class TrafficLight {
  private final int GREEN = 0;
  private final int ORANGE = 1;
  private final int RED = 2;

  private int showing;
  private int idNumber;

  public TrafficLight(                          ) {

  }
  public void setColour(int colourIndex) {
  }
  public void change() {

  }
  public int getId() {
  }
```

7

Ex01 - TrafficLight class

```
public String toString() {
      String outS = "";


      return outS;
 }

 public boolean showsSameColour(TrafficLight other) {

 }


 private String getColour(int index) {
    final String[] colours = { "GREEN", "ORANGE", "RED" };
    return colours[index];
 }
}
```

8

The word `static`

`static` is the keyword meaning that the **`static`** thing (constant, method or variable) is associated with a class (i.e. not associated with an instance of that class).

What does this mean in practice?

For a **`static`** constant, method or variable we do NOT need to first create an instance of the class before we can use it i.e. the **`static`** method, constant or variable belongs to the class.

9

`static` methods you have seen

Some examples of **`static`** methods which you have used:

```
double number = Math.random();
int bigger = Math.max(3, 5);
String valueS = Keyboard.readInput();
int value = Integer.parseInt(valueS);
```

10

`static` constants

Below is the TafficLight class definition. Look at the code on the next slide.

```
public class TrafficLight {
  private final int GREEN = 0;
  private final int ORANGE = 1;
  private final int RED = 2;

  private int showing;
  private int idNumber;

  public TrafficLight(int id) {...}
  public void setColour(int colourIndex) {...}
  public void change() {...}
  public boolean showsSameColour(TrafficLight o) {...}
  public String toString() {...}
  public int getId() {...}
}
```

11

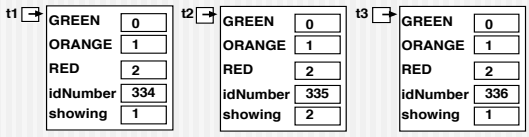
`static` constants

```
public class Program {
  public void start() {

   TrafficLight t1 = new TrafficLight(334);
   TrafficLight t2 = new TrafficLight(335);
   TrafficLight t3 = new TrafficLight(336);

   t1.setColour(1);
   t2.setColour(2);
   t3.setColour(1);

   if(t1.showsSameColour(t2) && t2.showsSameColour(t3)) {
         System.out.println("Same colour");
   }
  }
}
```

12

`static` constants

The three **`TrafficLight`** instances created on the previous slide look like this in memory:



13

`static` constants

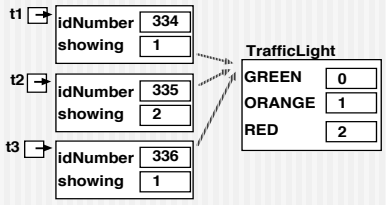
If I define the three constants as `static` there will only be ONE COPY of the three constants (`static` things are stored in the `TrafficLight` class).

```
public class TrafficLight {
  public static final int GREEN = 0;
  public static final int ORANGE = 1;
  public static final int RED = 2;

  private int showing;
  private int idNumber;

  public TrafficLight(int id) {...}
  public void setColour(int colourIndex) {...}
  public void change() {...}
  public boolean showsSameColour(TrafficLight o) {...}
  public String toString() {...}
}
```

14

`static` constants

The three `TrafficLight` instances from slide 11 now look like this in memory:



15

`static` constants

It is common to define constants as

`public static final ...`

Is it OK to define constants as `public`?

```
public class Program {
  public void start() {
    TrafficLight t1 = new TrafficLight(334);
    TrafficLight t2 = new TrafficLight(335);
    TrafficLight t3 = new TrafficLight(336);

      //We are able to refer to the static constants
    t1.setColour(TrafficLight.ORANGE);
    t2.setColour(TrafficLight.RED);
    t3.setColour(TrafficLight.ORANGE);
  } }
```

16

Ex02 - `equals()` method

Define the `equals()` method for the TrafficLight class:

```
public class TrafficLight {
  public static final int GREEN = 0;
  public static final int ORANGE = 1;
  public static final int RED = 2;

  private int showing;
  private int idNumber;

  public TrafficLight(int id) {...}
  public boolean equals(TrafficLight other) {



  }
  ...
}
```

17

Ex03 - Give the output

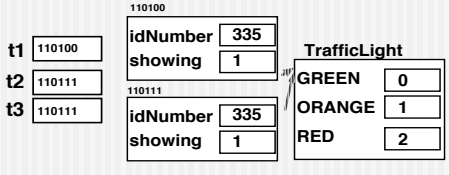
```
TrafficLight t1 = new TrafficLight(335);
TrafficLight t2 = new TrafficLight(335);
TrafficLight t3 = t2;

t1.setColour(TrafficLight.ORANGE);
t2.setColour(TrafficLight.RED);
t3.setColour(TrafficLight.ORANGE);

System.out.println("1. " + t1 == t2);
System.out.println("2. " + t1 == t3);
System.out.println("3. " + t2 == t3);

System.out.println("4. " + t1.equals(t2));
System.out.println("5. " + t1.equals(t3));
System.out.println("6. " + t2.equals(t3));
```

1.
2.
3.
4.
5.
6.

18

Ex03 - `equals()` method

A picture of the objects created in the code on the previous slide:



19

Passing primitive parameters

When we pass a parameter of a primitive type to a method a copy of the parameter is made and this copy is used inside the method.

```
public class Program {
 public void start() {
   int num1 = 15;
   method1(num1);
   System.out.println("num1: "+ num1);
 }
 private void method1(int num) {
    num = 33;
    System.out.println("num: "+ num);
  }
}
```

(1) start()
num1 15

(2) method1()
num 33

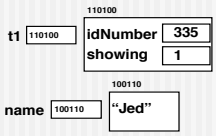
Inside `method1()`, the parameter, `num`, contains a copy of `num1`. Assigning 33 to the variable, `num`, does NOT effect the variable, `num1`, in the `start()` method.

20

Object variables - reference

Object variables hold a *reference* to an object of the type of the variable:

```
String name = "Jed";
TrafficLight t1 = new TrafficLight(335);
```



21

Passing object parameters

When we pass an object to a method as a parameter we make a copy of the REFERENCE to the object; we do not make a copy of the object itself.

This can lead to unexpected results.

22

Passing object parameters

```
public class Program {
 public void start() {
   TrafficLight t1 = new TrafficLight(335);
   t1.setColour(TrafficLight.RED);
   change(t1);
   System.out.println(t1.getColour());
}
private void change(TrafficLight t) {
   t.change();
} }
```

(1) start()
t1 110100

(2) change()
t 110100

110100
idNumber 335
showing 0

23

Ex04 - Give the output

```
public class Program {
 public void start() {
   TrafficLight t1, t2;
   t1 = new TrafficLight(335);
   t1.setColour(TrafficLight.RED);
   t2 = t1;
   method2(t2);
   System.out.println(t1.toString());
   System.out.println(t1 ==  t2);
 }
 private void method2(TrafficLight t){
   t.change();
 }
}
```

```
//output
```

NOTE: in `method1()`, `t` is a copy of the REFERENCE to the `TrafficLight` object. Only one `TrafficLight` object exists in memory.

24

Ex05 - Give the output

Arrays are objects.

```
public class Program {
 public void start() {
   int[] nums = {3, 4, 5, 6, 7};
   method3(nums);
   System.out.println(nums[2] + ", " + nums[1] );
 }

 private void method3(int[] a) {
    for(int i=0; i<a.length; i++) {
       a[i] = a[i] + 1;
    }
  }
}
```

25

Ex06 - Give the output

```
public class Program {
 public void start() {
   int[] nums = {3, 4, 5, 6, 7};
   nums = method4(nums);
   System.out.println(nums[2] + ", " + nums[3] );
 }
 private int[] method4(int[] a) {
    int[] b = new int[a.length];
    for(int i=0; i<a.length; i++) {
       b[i] = a[i];
    }
    a[2] = b[0];
    a[3] = 4;
    return b;
  }
}
```

26

Ex07 - Complete equalArrays()

```
public class Program {
 public void start() {
    int[] nums1 = {3, 4, 5, 6, 7};
    int[] nums2 = {3, 4, 5, 6, 7};
    int[] nums3 = {3, 4, 4, 6, 5};
    int[] nums4 = {3, 4, 4, 6};
    System.out.println(equalArrays(nums1, nums2));
    System.out.println(equalArrays(nums1, nums3));
    System.out.println(equalArrays(nums2, nums3));
    System.out.println(equalArrays(nums1, nums4));
 }
 private boolean equalArrays(int[] a, int[] b) {
             //returns true if the 2 parameter arrays have
             //elements with equal values, false otherwise


  }
}
```

```
true
false
false
false
```

27

What you need to know

Constants are defined as
`public static final ...`

The difference between equals() and ==

When passing object parameters you are passing the reference.

Arrays are objects.

28

TrafficLight class

```
public class TrafficLight {
       public static final int GREEN = 0;
       public static final int ORANGE = 1;
       public static final int RED = 0;

       private int showing;
       private int idNumber;

       public TrafficLight(int id) {
              idNumber = id;
              showing = (int)(Math.random() * 3);
       }
       public void setColour(int colourIndex) {
              showing = colourIndex % 3;
       }
       public int getId() {
              return idNumber;
       }
```

29

```
  public boolean showsSameColour(TrafficLight other) {
       if (showing == other.showing) {
              return true;
       }
       return false;
  }
  public String toString() {
       String outS = "";
       outS = "Traffic light: " + idNumber;
       outS = outS + ", showing: " + getColour(showing);
       return outS;
  }
  public void change() {
       showing = (showing + 1) % 3;
  }
  private String getColour(int index) {
       final String[] colours = {"GREEN", "ORANGE", "RED" };
       return colours[index];
  }
}
```