Lecture 3

We see some more *predefined* objects (colours) and we learn how to find out what their interfaces are; we discuss the idea of a **class** and a **class definition**, and we learn how to create *new* objects from an existing class definition.

We write a program that displays a window and changes its background colour repeatedly.

More predefined objects

- In Java (and in most of computing), colours are described by RGB values: three integers (0-255) specifying the amount of red, green and blue 'light'
- In Java, there are 13 predefined colour objects:

Name	R, G, B	Name	R, G, B
`white`	255, 255, 255	`yellow`	255, 255, 0
`orange`	255, 200, 0	`pink`	255, 175, 175
`magenta`	255, 0, 255	`red`	255, 0, 0
`lightGray`	192, 192, 192	`gray`	128, 128, 128
`darkGray`	64, 64, 64	`cyan`	0, 255, 255
`green`	0, 255, 0	`blue`	0, 0, 255
`black`	0, 0, 0		

- The full reference is, e.g., `java.awt.Color.red`, but we can arrange to use just, e.g., `Color.red`

Classes

- Objects fall into different categories, or **classes**
  - E.g. my bank account and yours belong to the same class: both are bank accounts
  - E.g. `Color.red` and `Color.cyan` both belong to the `Color` class
- We say that an object is an **instance** of its class
  - E.g. you are an instance of the student class
  - E.g. `Color.gray` is an instance of the `Color` class
- Objects in the same class have the same interface
  - they all have the same range of behaviours; they all offer the same set of services

The interface for `Color` objects

- We can find out the interface from on-line documentation. It includes `getRed(), getGreen()` and `getBlue()`
- How do we find out how much green light (0-255) there is in `magenta`?
- Putting `import java.awt.Color;` at the start of the file allows use of the shorter reference:
- How can we both find out and display the amount of green light in `magenta`?

Class definitions

- So far, we've used only predefined objects. What if they're not enough?
- E.g. there are, *in principle*, 256^3 different objects in the `Color` class, but only 13 are predefined
- For each class, Java has a **class definition**. A class definition is a kind of *'blueprint'* that specifies how to create new instances of that class
- We can ask Java to create a new object; it fetches the blueprint and builds the new object; the new object will have the same interface as other objects in the class because they were all built from the same blueprint
- Creating a new object from a class definition is referred to as **instantiation**

Constructing new `Color` objects

- Use the keyword `new` and supply 3 integers (RGB):
- But *typically* we also need to keep a *reference* to the object (its 'name'). Say we want to call it `purple`.
- Either:

  or

`JFrame`, another class definition

- Java has already got a class definition (blueprint) telling it how to construct windows, `JFrame`

- (Use `import javax.swing.JFrame;` for shorter references)

- To construct a new `JFrame` object with a given title, e.g.:

  ```
  JFrame f = new JFrame("A new window");
  ```

  (The window is initially invisible)

Part of `JFrame`'s interface

- `setBounds(x, y, w, h)` puts the frame's top-left corner at `x` pixels across and `y` down and sets its size to `w` pixels wide and `h` pixels high (where `x`, `y`, `w` and `h` are integers)

- `setVisible(true)` makes the frame visible; `setVisible(false)` makes it invisible

- `setBackground(c)` sets the background colour of the frame to `c` (where `c` is a `Color` object)

Example program

- We write a program that creates & displays a `JFrame` and repeatedly changes its background colour

- Although we'll cycle through many different `Color` objects, some colours will be approximated when they are displayed

- We must *either* use the full references *or* use `import`

- Observe that closing the window does not exit the program. We will have to interrupt the program using CTRL-C. We learn how to overcome this much later in the course.