Responsibilities

Two Sections of Code

- If we ask two people to work together to do a job, the first question to answer is:
- Who does what?
- We can only start to work out details of the individual tasks once we have divided up the responsibilities.

User Interface is Separate

- One very useful guideline is that the user interface should be separate from code that does the work.
- `main` can be responsible for the dialog with the user (user interaction).
- A helper object is then responsible for acting on the user’s wishes.

Drawing a Box

- There are 4 methods in the exercise 2 version.
  - `main`
  - `drawBox`
  - `drawTop`
  - `drawSide`
  - `drawRow` in exercise 3 version.
- Assigning responsibilities will govern the information flow: parameters and returns.

drawTop, drawSide, drawRow

- These responsibilities are easy to assign.
- Each draws one line of characters.
- They are given a `Console` and do not need to create it.
- They are given the number of characters on the line.
- `drawRow` is also told the end and middle characters.
- They do not return anything.

drawBox

- These responsibilities are harder to work out.
- Clearly it should draw the box!
  - It should use `drawTop` and `drawSide`.
  - They are helpers for the helper!
- This means that it should provide `drawTop` and `drawSide` with the information they need.
  - A `Console` for drawing
  - The width of the box.

drawBox (2)

- Should `drawBox` create the `Console` where the box is drawn?
- The spec says that each box is drawn on a new `Console`.
- So it is reasonable to ask `drawBox` to create this `Console`.
- It does not need a `Console` as input.

drawBox (3)

- Should `drawBox` get information from the user?
- NO. That would spread its responsibilities too far.
- So `drawBox` needs `nRows` and `nCols` as inputs.
- Some other part of the code should get this information from the user.

main

- The remaining task is interacting with the user.
- This should be done by `main`.
- It should control the loop, asking if the user wants to draw another box.
- It should also get the number of rows and columns for each box.

Main has two tasks?

- `main` does two things.
  - Gets the information for each box.
  - Controls the loop to draw many boxes.
- We could make a case that we need two methods, one for each of these tasks.
- However, both tasks are small and related.
- It makes sense to let `main` do both of them.
- Balance the number of different methods with their length.

Helper Methods Summary

- We define a helper method for each task.
- Each method does one thing well.
- User interaction is a separate task.
  - It should have a method dedicated to it.
  - That method is quite often `main`.
- Do not mix user interaction with other tasks.

Displaying Information

- Displaying information, such as drawing a box, is not user interaction.
- `drawBox`, `drawTop` and `drawSide` all display information.
  - They need a `Console` for the display.
  - They are not interacting with the user.
- Someone who talks all the time but does not listen is not interacting!

Static Objects

Unique Omnipresent Objects

- Sometimes we want to create a single helper object that is always there.
- All the methods of these sorts of object must be called `static`.
- The single unique object does not need to be created, it is always there.
  - Its name is the class name.
- Methods are called using this class name.

No Constructors

- They do not have constructors.
  - Instance variables can be initialised where they are defined.
  - See an example later.
- All main programs are `static` objects.

Java Static Objects

- `System.out` and `System.err`
- These are the names of the classes.
- Methods are called using these names.
  - `System.err.println(“Hello World”);`
- Java has many other `static` objects.

Memoriser Object

- This silly example is an object that memorises any number we tell it.
- It has 2 methods.
  - `public static void remember(int num);`
  - `public static int tellMe();`
- It stores the number in a `static` instance variable.
  - `private static int theNumber;`

The Code

```
public class Memoriser
  {
  private static int theNumber = 0;
  public static void remember(int num)
      {theNumber = num;}
  public static int tellMe()
      { return theNumber; }
  }
```

Mixed Objects

- Ordinary objects can have `static` members as well.
  - `static` methods
  - `static` instance variables.
- Ordinary methods can use `static` members.
  - The `static` object is always there.
- `static` methods can only use `static` members.

Current Time

- It would be useful if our `MyTime` class had a method the returned the current time.
- This would be a `static` method.
  - It must be always available.
- It can be implemented using the Java System object, which is also `static`.
  - It has a method that returns the current time counted in milliseconds since 1Jan 1970.
- We add this method to the existing ones.

Code

```
public static MyTime currentTime()
{
        // convert to seconds, divide by 1000
    int secs = System.currentTimeMillis() / 1000;
        // convert to mins, divide by 60
    int mins = secs / 60;
        // remainder divide by 1440 (mins in day)
    int mins %= 1440;
        // return object
    return new MyTime(mins / 60, mins % 60);
}
```

More on this

- `this` can be used inside methods.
- It is the object that called the method.
- `static` methods don’t have `this`.
- Instance variables can have `this` in front of them. It can make the code clearer.
  - `this.hours`.
- Or we can leave this out.
  - `hours`.

Reusing Constructors

- One constructor can call other constructors.
- They use the word `this` as the first line of the constructor.
- Parameters can be passed this way.
- `this` is useful if we want to provide default parameters to a constructor.

Console Constructors

```
public Console(String name, int nr, int nc)
{
  // the real constructor
}
  // provide default parameters
public Console()
  { this(“Console Window”, 30, 80); }
public Console(String name)
  { this(name, 30, 80); }
public Console(int nr, int nc)
  { this(“Console Window”, nr, nc); }
```