Helper Objects

Helper Object / Helper Method

- A helper ***method*** does a single action.
  - Draw a box, play a single game.
- And then it is finished.
- A helper ***object*** remembers information from one method call to the next.
  - The Console window and characters displayed in it.

Example: MyTime Objects

- A `MyTime` object stores a single time.
- It supports time based methods such as
  - Subtracting times
  - Adding times.
  - Converting times to and from strings.
  - Using 24 hour or 12 hour clocks.
- It can have many methods.

Can Be Used By Many Programs

- `Console` objects are helper objects
  - They are used by many programs.
- `MyTime` objects can also be used by many programs.
- This cuts down programming work
  - We can share code between many programs.
- The code only has to be tested once.

Expert In One Thing

- `MyTime` just knows about time.
- It does not know about dates or other related topics.
- This makes it easy to use in many different programs.
- It is not specialised.

Class and Objects

- All `MyTime` objects have the same properties.
  - The methods such as adding times.
- We define one `MyTime` class.
  - The Java code representing a time.
- Each `MyTime` object can store a different time.
- We create `MyTime` objects as we need them.
  - Using `new`.

Instance Variables

- Instance variables can be used by all methods of a `MyTime` object.
  - They have a wide scope.
- They are defined inside the class rather than inside a method.
- Methods can still define local variables.
  - They can only be used inside the methods in which they are defined.

Public Class and Methods

- The `MyTime` class is `public`.
  - It can be used by any other bit of code.
  - It is used when we create `MyTime` objects.
- The `MyTime` methods are `public`.
  - They can be called by any other bit of code.
- Methods are not `static`.
  - `MyTime` is not a unique, omnipresent object.

Private Data

- The instance variables are private.
  - They can only be used by methods in the `MyTime` class.
- We can only influence the instance variables by calling methods.
- This makes the code easier to understand and maintain.
- There are no hidden ways of changing the variables.

One class per file

- We need to create a new file called `MyTime.java`.
  - eclipse will do this automatically when we create a new class.
- In Java, each public class must be in a separate file.
- The filename must also match the class name.

MyTime – Instance Variables

```
public class MyTime
{
  private   int hours, mins;

  // methods go here
}
```

Initialisation - Constructors

- Objects must be initialised before being used, just like other variables.
- The instance variables must be given initial values.
- A constructor method does this.
- It is called when the object is created using `new`.
- Its name is the same is the class.
- It does not have a return type, not even `void`.

MyTime – Constructor

```
public class MyTime
{
   private    int hours, mins;
       // last lifetime of the object
       // each object has its own copy of them.

       // constructor
   public MyTime(int h, int m)
   {
       hours = h; // remember them before
       mins  = m; // they are destroyed
   } // h and m are destroyed here
}
   // example call
   MyTime a = new MyTime(1, 30);
```

Constructor Explained

- `hours` and `mins` are instance variables.
  - They exist as long as the object exists.
  - They can be used by any method.
  - They are private and cannot be used by main.
- `h`, `m` are parameters of the constructor.
  - Only last while the constructor is being called.
  - Long enough to initialise `hours` and `mins`.
  - They get their values from the constructor call.

Accessor Methods

- Accessor methods just return information about the object without changing it.
- We can have methods to return the hours and mins.
- By convention, accessors start with get.

```
    // accessors
public int getHours() { return hours; }
public int getMins()  { return mins; }
```

- Methods that change the instance variables are called **transformers**.

Different Internal Representation

- Private data means that we can change the instance variables without any customers noticing.
- They cannot get at private parts of a class.
- We can store a time as `totMins` (total minutes) instead.
- We rewrite the class internals, but not the method headings.
  - It looks the same to all customers.

MyTime – totMins

```
public class MyTime
{
   private    int totMmins;

       // constructor
   public MyTime(int h, int m)
   {
       totMins = 60 * h + m;
   }
   public int getHours() { return totMins / 60; }
   public int getMins()  { return totMins % 60; }
}
   // example call
   MyTime a = new MyTime(1, 30);
```

1030 to 10h, 30m

- Time can be represented by a single number of the form hhmm.
- MyTime can convert to and from this representation.
- Converting from hhmm to a MyTime object is constructing.
- Converting the other way is an accessor method.

hhmm Methods

```
public MyTime(int hhmm)
{
    hours = hhmm / 100;
    mins  = hhmm % 100;
}

public int getHhmm()
     { return 100 * hours + mins; }
```

More Than One Constructor

- We can have several constructors
  - Provided they have different parameters.
  - The compiler can work out which is which.
  - We must not forget one of the parameters or we will call the wrong constructor.
- They all have the same name: MyTime.

```
MyTime a = new MyTime(10, 30);
MyTime b = new MyTime(1030);
```

Adding and Subtracting Times

```
public MyTime sub(MyTime b)
{
    int totMins = 60 * this.hours + this.mins -
              60 * b.hours - b.mins;
    if (totMins < 0)
          totMins += 1440;
    return new MyTime(totMins/60, totMins%60);
}
```

this

- Note that the new object, the difference between the two times, is created using new.
- Adding MyTimes is very similar.
- We add two objects.
  - One is the parameter b.
  - The other is this. What is it?

Calling sub

```
MyTime x = new MyTime(10, 30);
MyTime y = new MyTime(2,17);
MyTime z = x.sub(y);
```

- The parameter `b` is clearly `y`.
- `this` is `x`, the object that is calling `sub`.
- We can't call it `x` inside `sub` because any `MyTime` object can call `sub`.
- `this` is another confusing Java word.