Lecture 8

We write another class definition, this one is for people's names. We will see that instance methods can often be divided into two kinds: *getters* and *setters*.

We will also re-consider how to initialise instance variables. We will see that we can write *constructors* that replace Java's default initialisation with our own initialisation.

What do we want the `PersonName` class to offer?

Some client code

```
PersonName myName = new PersonName();
myName.setTitle("Dr.");
myName.setForename("Derek");
myName.setSurname("Bridge");
System.out.println(myName.getInitials());
System.out.println(myName.getForename());

PersonName myAlias =  new PersonName();
myAlias.setTitle("Mr.");
myAlias.setForename("Derrick");
myAlias.setSurname("Bridges");
System.out.println(myAlias.getFullName());

myName.setTitle("Prof.");
myAlias.setTitle("Professor");

myAlias.setForename("Derick");
```


The `PersonName` interface

Getters ('accessors')

```
public String getTitle()
public String getForename()
public String getSurname()
public String getInitials()
public String getFullName()
public String getLastFirst()
```

Setters ('mutators')

```
public void setTitle(String aTitle)
public void setForename(String aForename)
public void setSurname(String aSurname)
```

(A useful class might offer many more.)

The `PersonName` class definition

```
/**
 * A class that represents people's names.
 * @author Derek Bridge 666 d.bridge@cs.ucc.
 */
public class PersonName
{
/* ====================================
       PUBLIC INTERFACE
   ====================================
*/

/* --Getters------------------------- */
   /**
    * Returns the title from this name.
    *
    * @return the title from this name.
    */
   public String getTitle()
   {

   }
```



```
   /**
    * Returns the forename from this name.
    *
    * @return the forename from this name.
    */
   public String getForename()
   {

   }

   /**
    * Returns the surname from this name.
    *
    * @return the surname from this name.
    */
   public String getSurname()
   {

   }
```



```
   /**
    * Returns the initials of this name.
    *
    * @return the initials of this name.
    */
   public String getInitials()
   {

   }

   /**
    * Returns the full name.
    *
    * @return the full name.
    */
   public String getFullName()
   {

   }
```

```
   /**
    * Returns the surname and forename
    * (in that order).
    *
    * @return the surname and forename.
    */
   public String getLastFirst()
   {

   }

/* --Setters------------------------ */
   /**
    * Sets the title in this name to the
    * specified title.
    *
    * @param aTitle the specified title in
    * this name.
    */
   public void setTitle(String aTitle)
   {

   }
```



```
   /**
    * Sets the forename in this name to the
    * specified forename.
    *
    * @param aForename the specified forenam
    * in this name.
    */
   public void setForename(String aForename)
   {

   }

   /**
    * Sets the surname in this name to the
    * specified surname.
    *
    * @param aSurname the specified surname
    * in this name.
    */
   public void setSurname(String aSurname)
   {

   }
```



```
/* ====================================
      INSTANCE VARIABLES
   ====================================
*/




}
```


Instance variable initialisation

- Instance variable initialisation (recap):
  - Java gives a default initialisation
  - Or, we can initialise them in the declaration
- Either way, we *so far* have to build the object and *then* set the its state to what we want:

```
PersonName myPseudonym = new PersonName()
myPseudonym.setTitle("Sir");
myPseudonym.setForename("Hugh");
myPseudonym.setSurname("Jeegoh");
```

- More convenient would be if we could supply values when we construct the object, e.g.:

```
PersonName myPseudonym =
  new PersonName("Sir", "Hugh", "Jeegoh")
```


Constructors

- Constructors are like methods,
  - but they have the same name as the class, as many parameters as you like, and no return type
- They are used to re-initialise the instance variables in an object when it is being built
  - use them when you want to supply actual parameters to be used for initialisation (or when initialisation is complex)

```
public PersonName(String aTitle,
   String aForename, String aSurname)
{


}
```


A constructor for `PersonName`

Add this to the start of `PersonName`, then revise the client (which can no longer use `new PersonName()`.)

```
/* ====================================
      CONSTRUCTORS
   ====================================
*/
   /**
    * Allocates a new PersonName object with
    * the given title, forename and surname.
    *
    * @param aTitle the title in this name.
    * @param aForename the forename in this
    * name.
    * @param aSurname the surname in this
    * name.
    */
   public PersonName(String aTitle,
      String aForename, String aSurname)
   {

   }
```