Lecture 7

We discuss, in detail, a third kind of Java variable, instance variables. We discuss the notions of variable declaration, initialisation, lifetime and identifier scope. Additionally, instance variables (and class variables) have a notion of visibility, which we discuss.

We re-implement the gibbering wreck class to give a `getRepetition` instance method.

Gibbering Wrecks again: the `getRepetition` instance method

```
/**
 * Returns whatever the wreck said
 * in whatever method it executed
 * previous to this one.
 *
 * @return a repetition of whatever
 * this wreck said previously
 */
public String getRepetition()
{
}
```

Why won't local variables or formal parameters suffice to store what was said previously?

Instance variables

- **Instance variables** allow an object to 'remember' something between one method invocation and another
- This is clearly essential:
  - e.g. if a wreck object is to remember what it said previously
  - e.g. if a bank account object is to remember the current balance in between transactions
  - e.g. if a window object (a `JFrame`) is to remember the width and height specified in a `setSize` method

Instance variables: declaration & initialisation

- Declaration: not within a method but at 'top-level' within a class definition

  ```
  private String previousUtterance;
  ```

- Initialisation: Java automatically initialises with a default value which depends on the variable's type, e.g.:

`int`	0	`boolean`	`false`
`double`	0.0	`String, Color,...`	`null`

  But you can initialise them yourself:

  ```
  private String previousUtterance =
     "I ain't said nothing yet!";
  ```

  Or you can re-initialise in a constructor (see later)

Instance variables: scope of identifier & lifetime

- Scope: suppose the variable is called `previousUtterance`
  - everywhere in this class definition, you can use `previousUtterance` and `this.previousUtterance`
  - in other class definitions, you must prefix with an object reference, e.g. `w1.previousUtterance` (*but you might still not be able to access the variable, depending on its visibility*)
- Lifetime: when, during program execution, the object is created, its instance variables are also created; when the object is destroyed, its instance variables are destroyed
- (So, when do objects get destroyed? At program termination, but sometimes sooner. See later)

Instance variables: visibility

- For an instance variable, there is the notion of **visibility**: which parts of the program are allowed to access it
- This is determined by the visibility modifier: `public, protected, private` or default (blank)
  - If `public`, the variable is visible to the whole program; this is totally unacceptable practice!
  - If `private`, the variable is visible only within this class definition; this is is the only good practice!
  - `protected` or default, see later
- This is not a security or privacy issue; it is about defining classes that are not unnecessarily interdependent (see later)

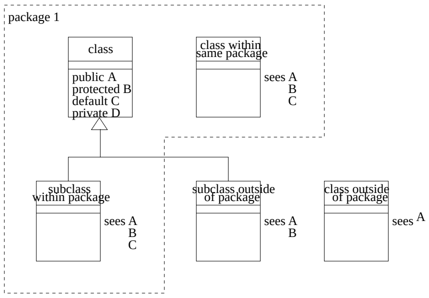
Where are the errors? Compile-time or run-time?

```
public class SomeClass
{
   private int v1;
   public int v2;
}

-------------------------------------

public class SomeClientClass
{
   public void someMethod()
   {  v2 = 3;
      SomeClass sc = new SomeClass();
      sc.v2 = 10;
      System.out.println(sc.v1);
   }
}
```


Visibility: summary



(From Coad,P., Mayfield, M. and Kern,J.: *Java Design: Building Better Apps & Applets (2nd edn)*, Yourdon Press, 1999)

The (almost) finished `Wreck` class definition

```
/**
 * A class that represents highly emotional
 * objects.
 * @author Derek Bridge 666 d.bridge@cs.ucc.
 */
public class Wreck
{
   /**
    * Displays a greeting on the standard
    * output.
    */
   public void displayGreeting()
   {

      System.out.println(
         "Hi. I’m a gibbering wreck.");
   }
```



```
   /**
    * Returns a greeting from this wreck.
    *
    * @return a string containing a greeting
    */
   public String getGreeting()
   {
      return "Hi. I’m a gibbering wreck.";
   }

   /**
    * Displays two random syllables on the
    * standard output. Each syllable compris
    * a random consonant-vowel pair.
    */
   public void displayGibberish()
   {  // Code omitted to save space.

      System.out.println(
                     syllable + syllable);
   }
```



```
   /**
    * Returns a copy of the parameter
    * concatenated to itself.
    *
    * @param aMessage a string that will be
    * mimicked.
    * @return the parameter concatenated to
    * an uppercase version of itself.
    */
   public String getMimicry(
                        String aMessage)
   {  String messageInUpperCase =
                  aMessage.toUpperCase();


      return aMessage + messageInUpperCase;
   }
```



```
   /**
    * Returns whatever the wreck said
    * in whatever method it executed
    * previous to this one.
    *
    * @return a repetition of whatever
    * this wreck said previously.
    */
   public String getRepetition()
   {

   }



}
```