Lecture 12

We will begin on another class definition, one for simple times. (Note that this is a useful example for the next few lectures. However, in practice, we need not write our own time class: the Java library already contains high-quality facilities for handling dates & times.)

We will discuss the keyword `final` and the use of named constants rather than 'magic numbers' or 'magic strings'.

This shows the second of the two major uses of class variables.

The `final` modifier

- When declaring any of Java's four kinds of variables, we can include the word `final` in the declaration
- After initialisation, the value of these variables can never be changed by the program. (Compiler error if you try subsequently to assign to them)
- Most commonly, use `final` on class variables (`static`) to give *named constants*
- In practice, instance variables are never `final`. Why?
- (`final` can appear elsewhere in programs too)

Avoiding magic numbers

- With only a few exceptions (e.g. some uses of 0, 1, 2, etc.), use expressions or named constants, not hard-coded magic numbers:

```
String str = "abcdefghijklmnopqrstuvwxyz"
for(int i = 0; i < 26; i++)
{   ... } // Bad

for(int i = 0; i < str.length(); i++)
{   ... } // Good

private LibraryBook booksOnLoan =
   new LibraryBook[6]; // Bad

private LibraryBook booksOnLoan =
   new LibraryBook[LOAN_LIMIT]; // Good
private static final int LOAN_LIMIT = 6;
```

- Avoid hard-coded `String`s, `char`s, etc. too

Advantages: readability

Compare this...

```
switch (month)
{  case 3: case 5: case 8: case 10:
   {  return 30;
   }
   case 1:
   {  if (isLeapYear(year))
      {  return 29;
      }
      else
      {  return 28;
      }
   }
   default:
   {  return 31;
   }
}
```

...with this...


```
switch (month)
{  case APR: case JUN: case SEP: case NOV:
   {  return SHORT_MONTH_LENGTH;
   }
   case FEB:
   {  if (isLeapYear(year))
      {  return LEAP_FEB_LENGTH;
      }
      else
      {  return NON_LEAP_FEB_LENGTH;
      }
   }
   default:
   {  return LONG_MONTH_LENGTH;
   }
}
public static final int JAN = 0;
public static final int FEB = 1; // etc.
public static final int
   NON_LEAP_FEB_LENGTH = 28;
public static final int
   LEAP_FEB_LENGTH = 29;          // etc.
```


Advantages: maintainability

Compare this...

```
private int[] marks = new int[6];
```

Elsewhere in your program:

```
int totalMark = 0;
for (int i = 0; i < 6; i++)
{  totalMark += mark[i];
}
return totalMark / 6;
```

Elsewhere again

```
for (int i = 0; i < 6; i++)
{  System.out.println(mark[i]);
}
```

...with this...

```
private int[] marks = new int[NUM_OF_EXAMS];

private static final int NUM_OF_EXAMS = 6;
```

Elsewhere in your program:

```
int totalMark = 0;
for (int i = 0; i < NUM_OF_EXAMS; i++)
{  totalMark += mark[i];
}
return totalMark / NUM_OF_EXAMS;
```

Elsewhere again

```
for (int i = 0; i < NUM_OF_EXAMS; i++)
{  System.out.println(mark[i]);
}
```

Consider what happens when the number of exams is changed

Named constants: `public` or `private`?

- Do you think clients would find the named constant useful or not?

```
public static final int JAN = 0;
private static final int NUM_OF_EXAMS = 6
```

- E.g. `public static final` variables in the Java library:

```
public static final double PI =
   3.14159265358979323846;
public static final int MAX_VALUE =
   2147483648;
public static final Color magenta =
   new Color(255, 0, 255);
```

- As we saw, clients have to prefix with the class name: `Math.PI`, `Integer.MAX_VALUE`, `Color.magenta` (after importing)

One client of the `Time` class

```
/**
 * A program that computes how long it
 * is until an appointment.
 * @author Derek Bridge 666 d.bridge@cs.ucc.
 */
public class HowLong
{
   public static void main(String[] args)
   {  Time now = new Time(14, 8, 25);
      System.out.println("Time now: " +
         now.getClockTime());

      Time appointment = new Time(16, 0, 0);
      System.out.println(
         "Time of appointment: " +
         appointment.getClockTime());

      System.out.println("Seconds to wait: "
         now.getIntervalInSecsUntil(
         appointment));
   }
}
```


Another client of the `Time` class

```
/**
 * Works out which  appointment is next.
 * @author Derek Bridge 666 d.bridge@cs.ucc.
 */
public class WhatNext
{  public static void main(String[] args)
   {  Time t1 = new Time(12, 0, 0);
      Time t2 = new Time(21, 30, 0);
      System.out.print(t1.getClockTime());
      if (t1.isBefore(t2))
      {  System.out.print(" before ");
      }
      else if (t2.isBefore(t1))
      {  System.out.print(" after ");
      }
      else // they're equal
      {  System.out.print(" equals ");
      }
      System.out.println(t2.getClockTime());
   }
}
```


Part of the `Time` class definition

(Comments essential but omitted to save space.)

```
public class Time
{
/* ====================================
      CONSTRUCTORS
   ====================================
*/ public Time(int theHour, int theMins,
      int theSecs)
   {

   }

/* ====================================
      PUBLIC INTERFACE
   ====================================
*/ /* ----Getters-------------------- */

   public boolean isEqualsTo(Time aTime)
   {
   }
```



```
   public boolean isBefore(Time aTime)
   {
   }

   public int getIntervalInSecsUntil(
      Time aTime)
   {
   }

   public String getClockTime()
   {
   }

/* ========================================
      INSTANCE VARIABLES AND CLASS VARIABLE
   ========================================
*/




}
```