Java

Writing instantiable classes

The tragedy of life doesn’t lie in not reaching your goal; the tragedy lies in having no goal to reach. Bill Newman

Lecture objectives

- We've been using predefined classes. Now we will learn to write our own classes to define new objects

- You should be able to understand
  - class declarations
  - instance variables and instance methods
  - encapsulation
  - visibility modifiers

Two types of Class

- Those not as templates for objects (no instances of these classes will be created):
  - Driver programs ie a class that contains a `main` method
  - Collections of constants and/or methods eg. `Math`

- Those used for defining and creating objects
  - instantiable classes
  - ie we can create instances of these classes (objects)

Instantiable classes

- An instantiable *class* is a blueprint used to create objects
  - many things or objects are made from a pattern or template
- It is a generalised case that defines what data we need to define a specific case (an object)
  - eg a car
    - What generalised data might we use to define cars?
    - What data would define a specific car object eg my car?
  - What about a book, a triangle, a student, a room…
- A class also defines methods to allow us to change objects

A prewritten instantiable class

- The `String` class is used to define `String` objects

```
String str;
```

- Each `String` object contains specific data (its state)

```
str = “a meaningless string”;
```

- Each `String` object can perform pre-defined methods

```
String newStr = str.toUpperCase();
```

Writing Classes

- Suppose we wanted to write a program that simulates the flipping of a coin
- Think of a coin *object* as an actual coin
- We could write a `Coin` class to define coin *objects*

- How would we define the state (data) of a coin?
- If you are just flipping a coin, what data would you need to define a coin?
- What method (or behaviour)could you apply to the coin?

Objects

- The state of the coin is its current `face` (heads or tails).
  - *How would you represent the face of a coin?*
- Because the state of an object can change, it is defined by variables ..known as *instance variables*
  - Each object is an instance of the Coin class
- The behaviour of the coin is that it can be flipped.
  - the behaviours of objects are defined by *instance methods*
  - *How would you represent flipping a coin?*
- Note that the behaviour of the coin might change its state

Using instantiable classes

- Write a driver program
  - with a `main` method
  - (processing starts here)
  - calls the class constructor to create an object
  - utilises the object's methods
  - (processing ends)

- Write a class definition
  - define instance variables
  - define a constructor
  - define instance methods
    - to flip the coin
    - to return instance variable

```
//   Driver program
public class CoinFlip
{
     public static void main (String[] args)
     {
         Coin  myCoin = new Coin();

          myCoin.flip();


           int result = myCoin.getFace();

          // 0 = heads, 1 = tails
          if (result == 0)
              System.out.println ("It's a head!");
          else
              System.out.println ("It's a tail!");
     }
}
```

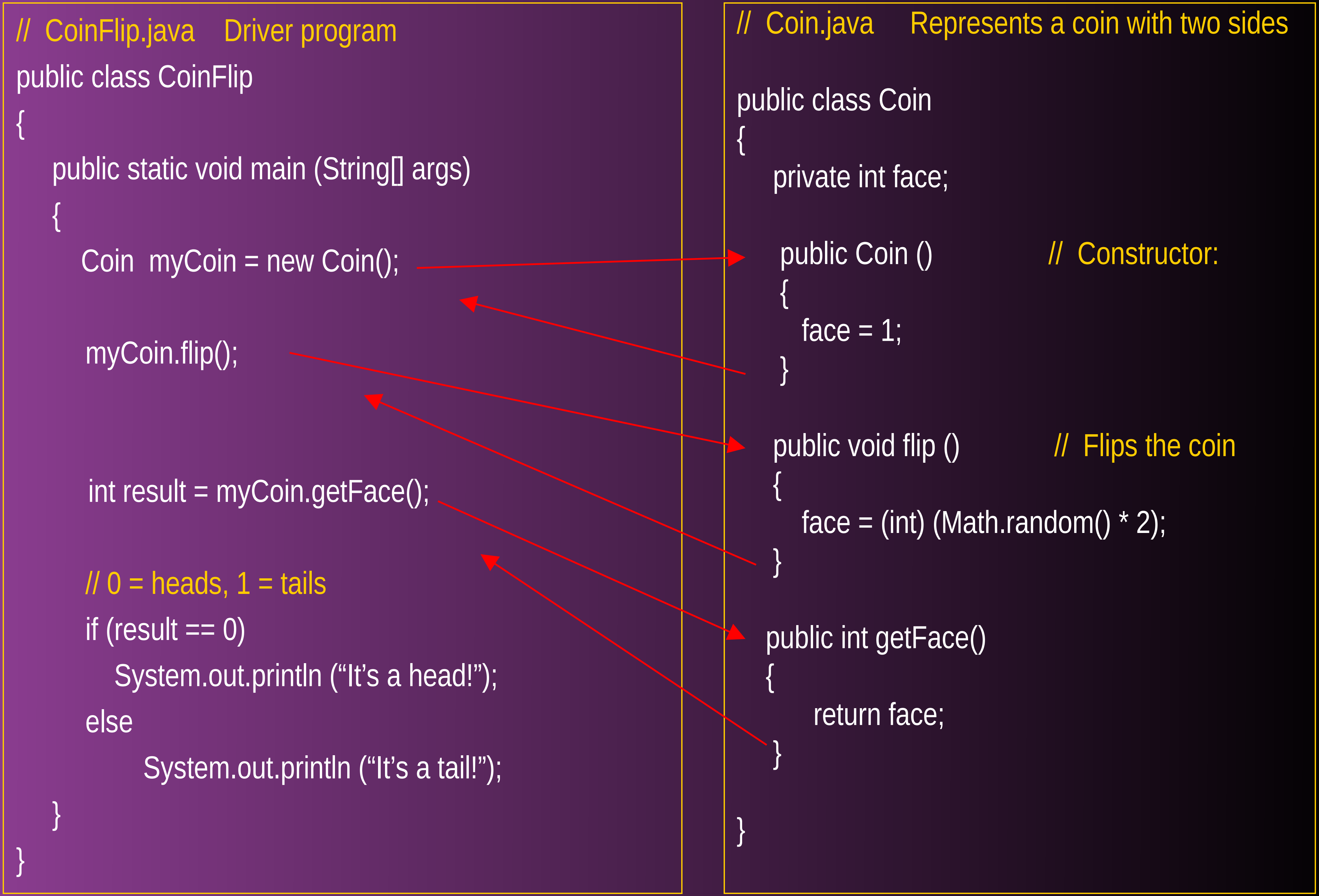
public class Coin

Creates an object of the Coin class called `myCoin`

Invokes the `flip` method of the Coin class and applies it to the object

Invokes the `getFace` method to access the instance variable `face`

```
//  Coin.java     Represents a coin with two sides

public class Coin
{
     private int face;


      public Coin ()                 //  Constructor:
      {
         face = 1;                   //  initialise
      }


     public void flip ()             //  Flips the coin
     {
          face = (int) (Math.random() * 2);
     }


     public int getFace()
    {
          return face;
     }


}
```



```
//  CoinFlip.java    Driver program
public class CoinFlip
{
     public static void main (String[] args)
     {
         Coin  myCoin = new Coin();

         myCoin.flip();


         int result = myCoin.getFace();

         // 0 = heads, 1 = tails
         if (result == 0)
             System.out.println ("It's a head!");
         else
                 System.out.println ("It's a tail!");
     }
}
```

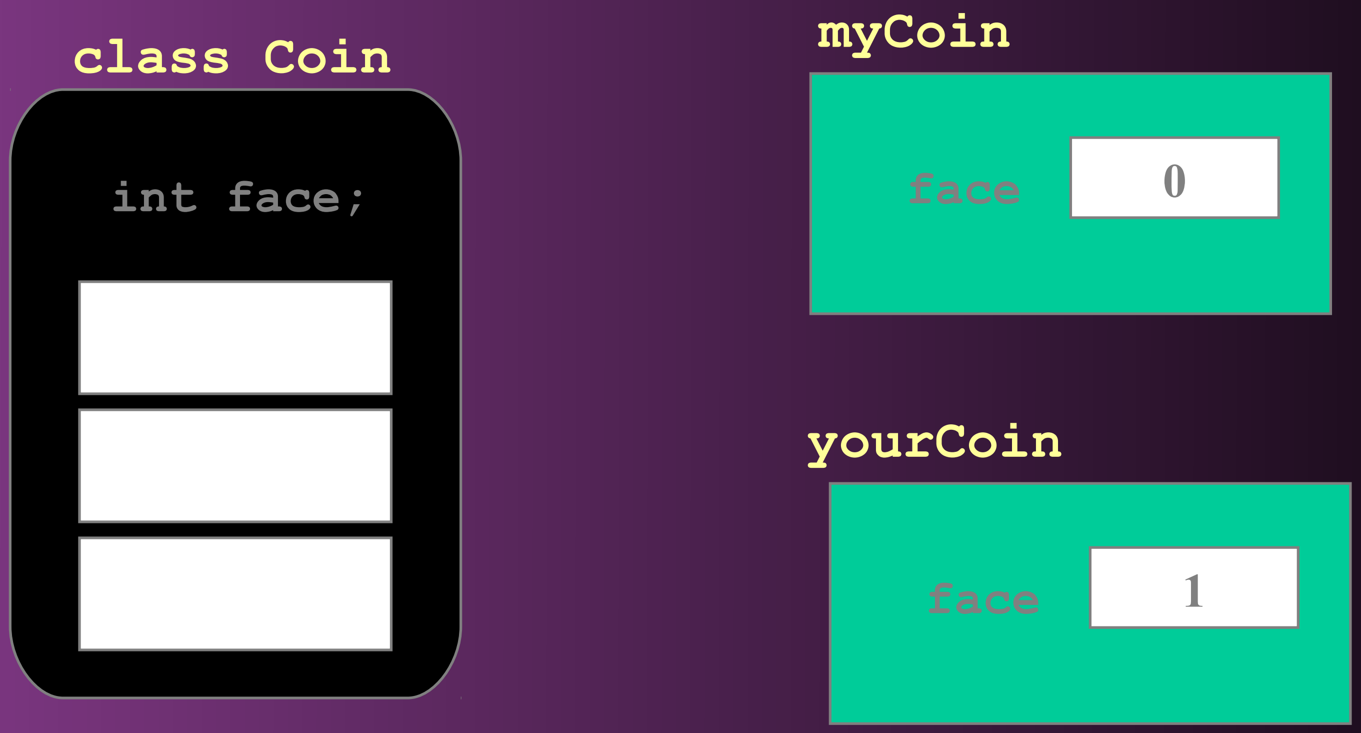
```
//  Coin.java     Represents a coin with two sides

public class Coin
{
     private int face;

      public Coin ()                //  Constructor:
      {
         face = 1;
      }

     public void flip ()            //  Flips the coin
     {
         face = (int) (Math.random() * 2);
     }

    public int getFace()
    {
          return face;
    }

}
```

```
//  CountFlips.java    Driver program counts heads
public class CountFlips
{
    public static void main (String[] args)
    {
       int result, myScore = 0, yourScore = 0;
       Coin  myCoin = new Coin();
       Coin yourCoin = new Coin();

       for (int i = 1; i <= 10; i ++)
       {
           myCoin.flip();
           result = myCoin.getFace();
           if (result == 0)
               myScore = myScore + 1;

           yourCoin.flip();
           result = yourCoin.getFace();
           if (result == 0)
               yourScore = yourScore + 1;
       }
       System.out.print(“me “ + myScore + “you “ +
                                            yourScore;
    }
}
```

```
//  Coin.java     Represents a coin with two sides

public class Coin
{
    private int face;

     public Coin ()               //  Constructor:
     {
        face = 1;
     }

    public void flip ()           //  Flips the coin
    {
        face = (int) (Math.random() * 2);
    }

   public int getFace()
   {
         return face;
    }

}
```

Instance Data

CountFlips.java



class Coin

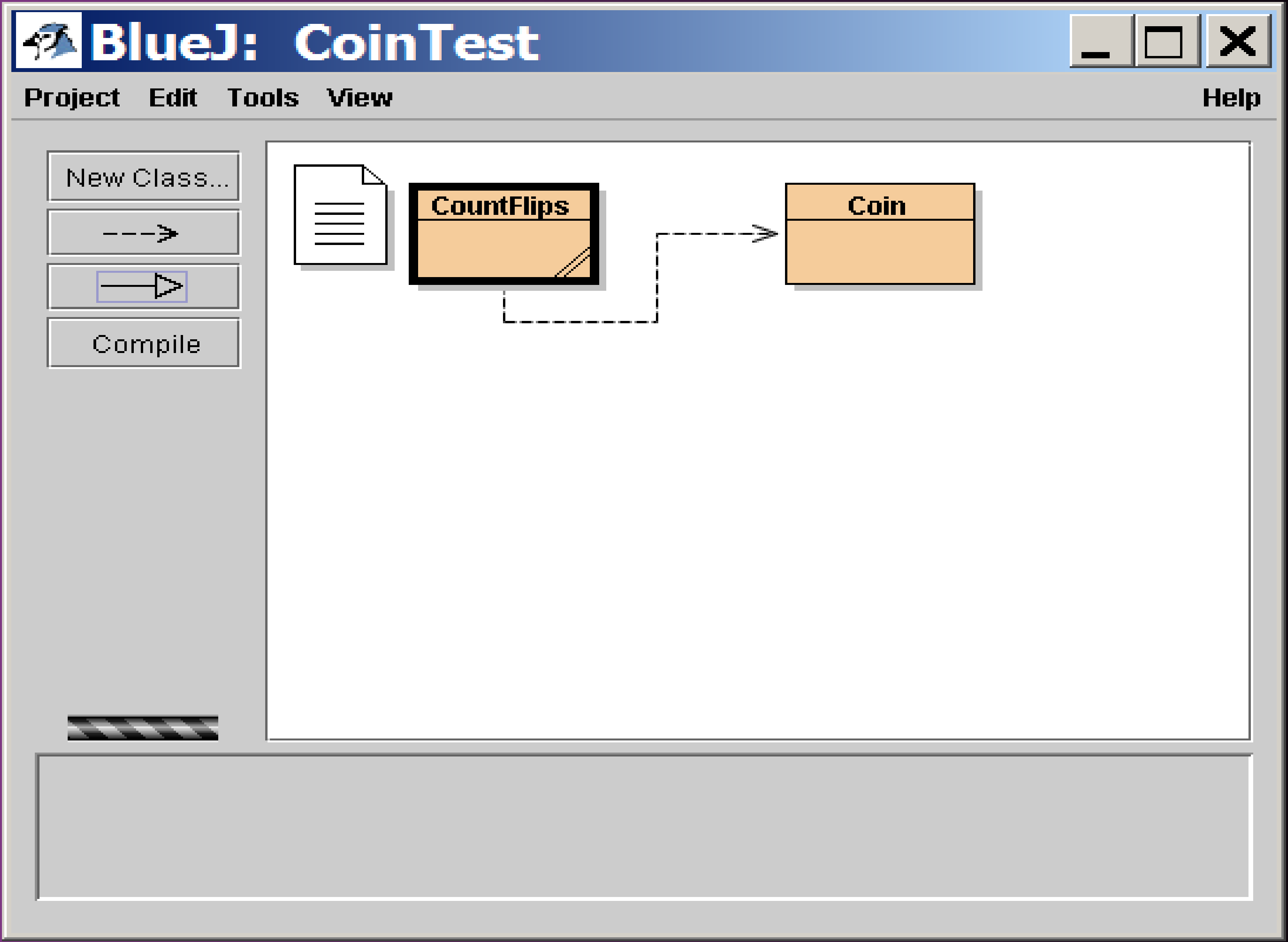
int face;

myCoin

face 0

yourCoin

face 1


BlueJ: CoinTest
Project
Edit
Tools
View
Help
New Class...
Compile
CountFlips
Coin

Practice

- Write an instantiable class called Counter that has
  - An integer instance variable called *numClicks*
  - A constructor that sets *numClicks* to 0
  - A method called click() that adds 1 to *numClicks*
  - A method called getClicks() that returns *numClicks*

- Write a driver program called TestCounter that
  - Creates a counter object
  - Calls the click method 3 times
  - Calls the getClicks() method and prints its value

```
public class TestCounter
{
    public static void main (String [] args)
    {
         //create Counter object



        // call click method



        //get number of clicks



        // print number of clicks


    }
}
```

```
public class Counter
{
    define instance variable


    define constructor
    {
    }


    define click method
    {
    }



    define get number of clicks method
    {
     }
}
```

Another example

- Consider creating a Date class
- It accepts a date as 3 integers in the format

```
dd, mm, yyyy
```

- It formats the input date so that it is output as follows:

dd/mm/yy   eg 03/09/03

```
// TestDate.java
public class TestDate
{
        public static void main (String [] args)
        {
            String displayDate;

            Date sem1 = new Date();
            Date sem2 = new Date();

            sem2.setDate(26,7,2003);

            displayDate = sem1.getDate();
            System.out.println(displayDate);

            displayDate = sem2.getDate();
            System.out.println(displayDate);
        }
}
```

```
The date is 01/01/00
The date is 26/07/03
```

```
import java.text.*;
public class Date
{
        private int day;
        private int month;
        private int year;
        public Date()
        {
                day = 1;
                month = 1;
                year = 2000;
        }
        public void setDate(int dd, int mm, int yyyy)
        {
                day = dd;
                month = mm;
                year  = yyyy;
        }
        public String getDate()
        {
               DecimalFormat df = new DecimalFormat (“00”);
             String dateString = (“The date is “ + df.format(day) +
                    ‘/’ + df.format(month) + ‘/’ + df.format(year%100) );
             return dateString;
        }
}
```

Constructors

- A constructor is a special method that contains instructions to set up a newly created object
- When writing a constructor, remember that:
  - it has the same name as the class
  - it is syntactically similar to a method
  - it does not return a value
  - it has no return type, not even `void`
  - it often sets the initial values of instance variables
  - it is invoked by the keyword `new`

```
Coin  myCoin = new Coin();
```

```
public Coin ()
{
    flip();
}
```

Passing data to a Constructor

- The *Coin* or *Date* constructors did not require any data to be passed to it them create objects
  - the instance variables were initialised by existing data
- Often constructors require data to initialise objects uniquely

- Note: if no constructor is provided Java will provide a default one to create an object
  - instance variables will be set to default values

Passing data to a constructor

- Consider an Account class for a bank
- What instance variables could define each object?
- What methods would the class require?

- When setting up a new account (ie creating an Account object), it would be useful to create it and initialise the instance variables appropriately
- The constructor accepts parameters just like methods

```
//  TestAccount.java
public class TestAccount
{
public static void main (String[] args)
   {
      Account acct1 = new Account (“J Bond", 72354, 102.56);
      Account acct2 = new Account (“M Munro", 69713, 40.00);

      acct1.deposit (25.85);
      acct2.deposit (500.00);
      double currentBal = acct1.getBalance();
      System.out.println (“acct1 balance: " +   currentBal);

      acct2.addInterest();
      currentBal = acct2.getBalance();
      System.out.println (“acct2 balance: " +   currentBal);

      System.out.println (acct1);
      System.out.println (acct2);
    }
}
```

```
//  Account.java
public class Account
{
   private final double RATE = 0.045;  // interest rate 4.5%
   private int acctNumber;
   private double balance;
   private String name;

   public Account (String owner, int account, double initial)
   {
       name = owner;
       acctNumber = account;
       balance = initial;
   }
   public void deposit (double amount)
   {
       balance = balance + amount;
    }

  public double getBalance()
  {
      return balance;
  }

   public void addInterest ()
   {
     balance = balance + (balance * RATE);
   }
```

Visibility (or Access) Modifiers

- Usually the declaration of
  - an instance variable,
  - a constructor, or
  - an instance method

begins with an *visibility modifier* (`public` or `private`)

- A *visibility modifier* determines whether that entity can be
  - accessed by other classes (`public`) or
  - accessed only by methods within the class itself (`private`)

Visibility

- The most common arrangement is for *instance variables* to be `private`
- This makes access to them available only by methods within the class

```
private int face;
```

Access to instance data

- The only access to `face` therefore will be through the instance methods provided in the `Coin` class

```
private int face;

public void flip ()             //  Flips the coin by randomly choosing a face
{
   face = (int) (Math.random() * 2);
}

 public int getFace ()       //  Returns true if the current face of the coin is heads
{
  return face;
}
```

- The driver program cannot access the `face` variable directly

Visibility – constructors and methods

- *Constructors* and *methods* that provide the object's services are usually declared with `public` visibility
  - Then they can be invoked by clients

- So any program that uses a `Coin` object can invoke the following:

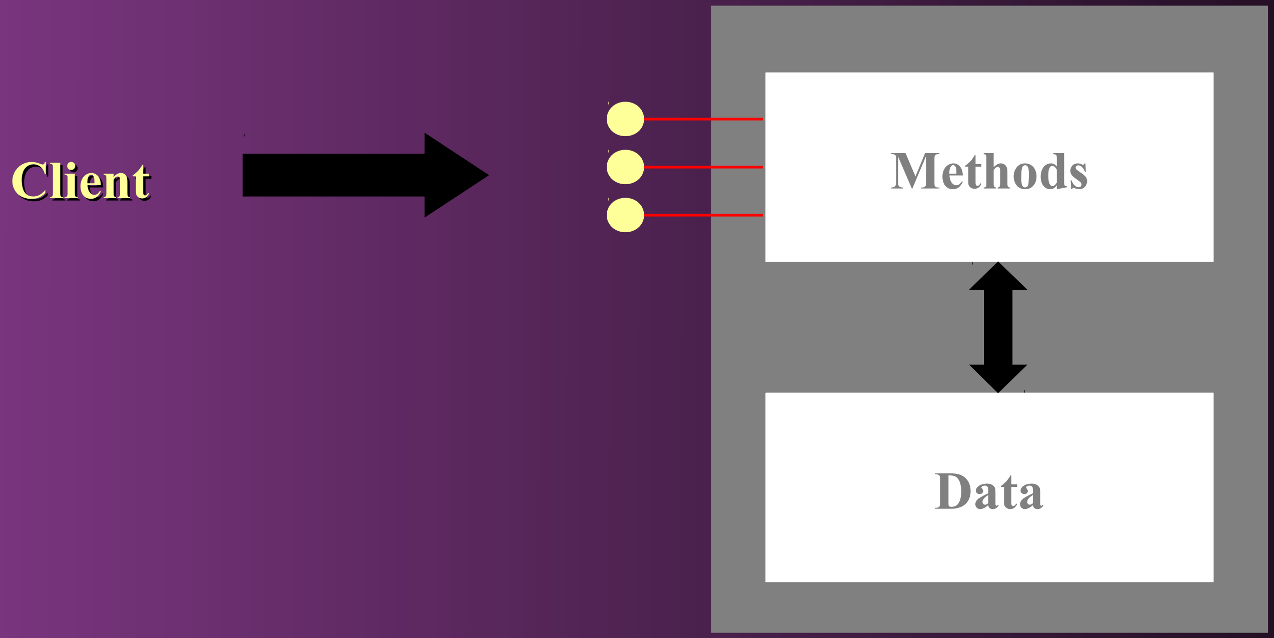
```
public Coin()                    //constructor

public void flip()               //method

public int getFace()             //method
```

Information hiding

- By limiting access to the variables in a class
  - making them `private`
- If access to a variable is needed outside the class,
  - provide a method that returns the value of the variable
  - and/or a method that changes the value of the variable.
- Methods that modify the variables can check the validity of the new values

Encapsulation

- An encapsulated object can be thought of as a *black box*
- Its inner workings are hidden to the client, which only invokes the methods



Documentation

- Most documentation has been removed from the programs shown on the lecture slides to conserve space
- Do not treat these programs as being sufficiently documented – they are not!
- Now that you are developing methods it is particularly important that every class and every method is documented, as in the text

```
//----------------------------------------------------------------
//  Sets up the coin by flipping it initially.
//----------------------------------------------------------------
public Coin ()
{
   flip();
}
```

Lecture Outcomes

Today we have covered:

- writing your own instantiable classes
- creating objects from them
- we have moved into true object-oriented programming

- Questions?