Constructors

- Methods called when the object is created
- Useful for initialisation
- Syntax
  - Name is the same as the class
  - No return type

Constructors (cont.)

```
public class Button
  { ...
    public Button ()
      { release();
      } // constructor

    private void displayBitmap ()
      {...}
    public void press ()
      {...}
    public void release ()
      {...}
  } // class Button
```

Constructors (cont.)

- Constructors can also:
  - take parameters
  - be overloaded

```
public Button (int initState)
  { if (initState == UP)
      release();
    else
      press();
  } // constructor
```

Using Constructors

- First constructor:

```
Button b1 = new Button();
```

Parameters for the constructor

- Second constructor:

```
Button b2 = new Button(Button.UP),
       b3 = new Button(Button.DOWN);
```

Static Class Members

- Most often class members are *instance members*
  - Each object (*instance*) has its own members
    - E.g. **`b1.visible`** is different to **`b2.visible`**

- Sometimes objects must share data/methods
  - The data/methods belong to the *class*
  - Use **`static`**

Static Data Example

- Example: Objects that need a unique serial number
  - Each object has an instance variable for its own serial number
  - All objects share a counter to keep track of the next serial number

Serial Number Example

```
public class MyObject
  { private static int nextNumber = 1;
    private int serialNumber;

    public MyObject ()
      { serialNumber = nextNumber;
        nextNumber = nextNumber + 1;
      } // constructor

    public int getSerialNumber ()
      { return serialNumber;
      } // getSerialNumber

  } // class MyObject
```

Serial Number Example (cont.)

```
MyObject o = new MyObject(),
         o2 = new MyObject(),
         x = new MyObject();
```

Static Method Example

- Methods can also be static
  - Belongs to class, not to an object
  - Useful for “utility functions”

Example of Static Methods

```
class MathematicalOps
  { public static double squareRoot (double x)
      { // Calculate the square root of x
        ...
      } // squareRoot

    public static double sin (double x)
      { // Calculate and return the sine of x
        ...
      } // sin
    ...
  } // class MathematicalOps
```

Example of Static method (cont.)

- Can use static methods without creating any object(s):

```
double x, y, angle, hypotenuse;
...
hypotenuse =
    MathematicalOps.squareRoot(x * x + y * y);
...
double s = MathematicalOps.sin(angle);
```

Use class name to call static method

Static Members - Notes

- The **`Math`** class already contains the mathematical functions
- We can use static data/methods without creating an instance of the class
  - Use the class name as a “reference”
- Static methods *cannot* use non-static data

Last Words

- Why is the **`main`** method always **`public`** and **`static`**?
  - **`public`** so that the JVM can see it
  - **`static`** so that the JVM can call it without creating an object from our “program class”

Chapter 7
Arrays

Basic Concepts

- Declaring an array:

```
type identifier [];
```

Or:

```
type[] identifier;
```

Example:

```
int myArray[];
```

Creating an Array

- Arrays are *objects*
- To create the array we use **`new`**

```
myArray = new int[10];
```

Working with an Array

- Usual subscripting operation supported

```
myArray[0] = 33;
myArray[1] = 25;
myArray[9] = myArray[0] + 1;
x = myArray[0] * 2 + myArray[1];
System.out.println(”Ans = ” + myArray[1]);
```

- Arrays *are* checked for subscript errors

Declaring Arrays

`int myArray[];` Or `int[] myArray;`

- What is the difference?

Nothing!

- But

`int a, b, c[];` `int[] a, b, c;`

A BIG difference!