Variables

Lifetime, Scope, Categories,
Shadowing and Composition

Variables

- A ***variable*** is a named storage location in the computer's memory.
- This named location or **variable** is used to store data while the program is running.
- Where a variable is declared has an effect on where it can be accessed / used.

Lifetime of variables

- In Java, the *garbage collector* periodically reclaims objects when they are no longer used
- This may be when a method finishes running.
- Variables declared within a method only exist within that method. They are only created when the method is called, and they only exist for as long as the method is running.

Lifetime of variables

```
public static double calculateArea(double radius)
   {
      double area = 3.147 * radius * radius;
      return area;
   }
```

- Here, `area` is a local variable. It only exists within the method `calculateArea`.
- `area` must be initialised within the method.
- It is created when the method is called, and ceases to exist when the method finishes running.

Lifetime of variables

```
public static double calculateArea(double radius)
  {
     double area = 3.147 * radius * radius;
     return area;
  }
```

- Here, `radius` is known as a parameter variable. Again, it only exists within the method `calculateArea`.
- `radius` is initialised when the method is called. It ceases to exist when the method finishes running.

Variable Scope

- Variables are only “visible” within the block of code in which they have been declared

  (within the pair of curly brackets)
- If they are referred to in another part of the program will cause a compiler error
- ***Local variable*** - Variable declared inside a method – *local* to that method
- A method is unaware of variables declared in any other method

Consider the code…

```
public class ScopeExample1For
{
  public static void main(String[] args)
  {
     for(int i = 0; i<5; i++)
     {
          System.out.print("*");
     }
     System.out.println(i);
  }
}
```

Here, i is local to the for loop, so it can't be accessed outside of that loop.

What's the problem here?

```
// returns the area
     public static double calculateArea(double width)
     {
            double area;
            area = width * width;
            return area;
     }
// a stupid method to illustrate a point
     public static void otherMethod()
     {
            System.out.print(area);
     }
```

Is this a problem?

```
public static int findLargest(int [] array)
{
   int largestYet = array[0];
   for(int i=0; i<array.length; i++)
   {
      if(array[i] > largestYet)
      {
         largestYet = array[i];
      }

   }
   return largestYet;
}

public static int countInArray(int [] array, int value)
{
   int count = 0;
   for(int i=0; i<array.length; i++)
   {
      if(array[i] == value)
      {
         count++;
      }

   }
   return count;
}
```

Is this a problem?

```
public static int findLargest(int [] array)
{
   int largestYet = array[0];
   for(int i=0; i<array.length; i++)
   {
      if(array[i] > largestYet)
      {
         largestYet = array[i];
      }

   }
   return largestYet;
}

public static int countInArray(int [] array, int value)
{
   int count = 0;
   for(int i=0; i<array.length; i++)
   {
      if(array[i] == value)
      {
         count++;
      }

   }
   return count;
}
```

Is this a problem?

```
public static void main(String [] args)
{
   int number = 5;
   System.out.println(number);
   printNumber(3);
   displayStars();
}

public static void printNumber(int number)
{
   System.out.println(number);
}

public static void displayStars()
{
   int number = 5;
   for(int i=0; i<number; i++)
   {
      System.out.print("*");
   }
}
```

Is this a problem?

```
public static void main(String [] args)
{
   int number = 5;
   System.out.println(number);
   displayStars();
}


public static void displayStars()
{
   for(int i=0; i<number; i++)
   {
      System.out.print("*");
   }
}
```

Scope vs Lifetime

- Scope of a variable is the set of lines of code from where you can refer to it – or “see” it
- Lifetime of a variable is the time from creation to the time of deletion.

Categories of Variables

- Categories of variables
  - Instance variables (`width` in `Oblong`)
  - Local variables (`area` in `calcArea()` method)
  - Parameter variables (`widthIn` in `setWidth()` method)
- An instance variable belongs to an object

```java
public class Oblong{
   // instance variables
   private double width;
   private double height;

   // the methods
   public void setHeight(double heightIn)
   {
        height = heightIn;
   }

   // returns the area of the oblong
   public double calculateArea()
   {
        double area;
        area = width * height;
        return area;
   }
}
```

Instance variable

Parameter variable

Local variable

Shadowing Instance variables

- This is when a local or parameter variable is declared with the same name as an instance variable.
- Java allows this, but it causes problems. It is bad programming practice and should be avoided.
- Java will associate the name with the local variable instead of the instance variable.

Shadowing instance variables

```
public class Oblong{
  // instance variables
  private double width;
  private double height;
  :
  :
public void setWidth (double width)
{
   width = width;
}
```

This causes the instance variable width to stay the same. The parameter variable width is assigned the value of itself. This is pointless.

Solution

```
public class Oblong{
  // instance variables
  private double width;
  private double height;
  :
  :
public void setWidth (double widthIn)
{
   width = widthIn;
}
```

Continued…

Shadowing instance variables

```
public class BankAccount{
  // instance variables
  private double balance;

  public void deposit (double balance)
  {
   balance = balance + balance;
  }
```

Again, this will compile. However, it will produce unexpected results.

Composition

- Composition is where some or all of an Object’s instance variables are themselves Objects.
- For example, a Student class may have a name, which is a String.
- Composition represents a “has a” relationship in Java.
- This is another form of code reuse.

Composition Example

```java
// class representing an electronic Book
import java.util.ArrayList;
public class EBook
{
   // instance variables
   private String title;
   private int sizeInMegabytes;
   private ArrayList <String> authors;

   public EBook(String titleIn, int sizeInMegabytesIn)
   {
      title = titleIn;
      sizeInMegabytes = sizeInMegabytesIn;
      authors = new ArrayList <String>();
   }
```

Composition Example

```java
   public void addAuthor(String authorIn)
   {
      authors.add(authorIn);
   }

   public void printAuthors()
   {
      for(String s: authors)
      {
         System.out.print(s + " ");
      }
   }
```