

# JAVA ASSIGNMENT

## Task 3

1. ~~Explain~~ Differences between primitive and reference data types.
- Primitive data types store the actual values whereas reference variables store the addresses of the objects they refer to.

2. Define scope of a variable (local and global variable)

- A scope is a region of the program ~~and broadly speaking~~

### Local variables

- Variables are declared inside a function or block are local variables.

### Global variables

- Variables that are defined outside of all the functions usually at the top of the program. It'll hold their type throughout the life-time of your program.

3. Why is initialization of the variables required?

Because unless the variable has static storage space, its initial value is indeterminate. You cannot rely on it being anything as the standard does not define it.

4. Differentiate between static, ~~not~~ instance and <sup>local</sup> ~~global~~ variables:

- L.V - Defined within a method or a code block

- I.V - Defined outside the method at the class level

- S.V - Defined outside a method at the class level.

- L.V - Is only accessible in the method / code block where it is declared

- I.V - Is accessible throughout the class

- S.V - Is accessible throughout the class

- L.V - Remains in memory as long the method executes.
- I.V - Remains in memory as long as the object is in ~~meth~~ <sup>memory</sup>.
- S.V - Remains in memory as long as program executes.

##### 5 Differentiate between widening and narrowing casting in java.

- Widening casting (smaller to larger type) <sup>target</sup>  
Can happen if both types are compatible and ~~larger~~ type is larger than source type. Takes place when two types are compatible and the target type is larger than the source type.  
Widening casting (smaller to larger type)  
Can happen if both types are compatible and the target type is larger than source type. Takes ~~place~~ <sup>place</sup> when two types are compatible and the ~~target~~ type is larger ~~than~~ than the source type.

- Narrowing casting (larger to smaller type).  
When we are assigning a larger type to a smaller type, Explicit Casting is required.  
Narrowing a class type  
When we are assigning larger type to a smaller type, then we need to explicitly typecasting it.

##### 7. Importance of using java packages:

They are used to group classes. We use packages to avoid name conflicts and to write a better maintainable code.

##### 8. Explain 3 controls used when creating GUI applications in java language:

- Java BorderLayout
- A BorderLayout places components in up to five areas; top,



bottom, left, right and centre. It is the default layout manager for every java JFrame.

#### - Java BorderLayout

A BorderLayout places components in up to five areas: top, bottom, left, right and centre. It is the default layout manager for every java JFrame.

#### - Java FlowLayout

FlowLayout is the default layout manager for every JPanel. It simply layout components in a single row one after the other.

#### - Java GridLayout

It is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell.

#### 9. Differences between containers and components:

- Java components class represents visual elements of a Graphical User Interface. (Its subclasses include JButton, JCheckBox, JTextfield, JChoice and JCanvas).

- The container subclass is another subclass of component. A container is a component that can contain other components (including other containers).

10. Write a java program to reverse an array having five items of type int.

```
import java.util.Scanner;  
import java.util.stream.StreamScanner;  
public class Main {  
    public static void main (String [] args) {  
        Integer [] intArray = { 10, 20, 30, 40, 50 };
```

```
// Print array starting from the last element  
System.out.println ("Numbers printed in reverse:");  
for (int i = intArray.length - 1; i >= 0; i--)  
    System.out.print (intArray [i] + " ");  
}
```

Output;

Numbers printed in reverse:

50, 40, 30, 20, 10.

11. Program written for GUI have to deal with "events".

Explain what is meant by the term ~~an~~ event.

An event is anything that can occur asynchronously, not under the control of the program to which the program might want to respond.



Give two different examples of events and discuss how a program might respond to these events.

In many (but not all) cases, an event is the result of a user action, such as when the user clicks the mouse button, types a character or clicks a button. The program might respond to a mouse click on a canvas by drawing a shape, to a typed character by adding the character to an input ~~box~~ box, or to a click on a button by clearing a drawing.

12. Explain difference of the following terms:

Polymorphism and encapsulation

- Polymorphism is another fundamental concept. It allows us to use a single interface with different underlying forms such as data types or classes.
- Encapsulation is one of the fundamental concepts in OOP. It describes the idea of restricting access to methods and attributes in class.

Method overloading and method overriding

- Method overloading is used to increase the readability of the program.
- Method overriding is used to provide the specific implementation of the method that is already provided by the super class.

Class and Interface

- A class can extend only one class but can implement any number of interfaces.
- An interface can extend any number of interfaces but cannot implement any interface.



## Inheritance and Polymorphism

- Inheritance allows us to define a class that inherits all the methods and attributes from another class.
- Polymorphism - allows us to use single interface with different underlying forms such as data types or ~~class~~ class.

6. Complete the following table filling in the missing values.

| Type    | Size<br>(In bytes) | Default | Range                        |
|---------|--------------------|---------|------------------------------|
| Boolean | 1 bit              |         | True,<br>False               |
| Char    | 2                  |         | '\0000' to<br>'\uffff'       |
| Byte    |                    | 0       | -27 to<br>+27-1              |
| Short   |                    | 0       | -215 to<br>+215-1            |
| Int     | 4                  |         | -231 to<br>+231-1            |
| Long    |                    | 0L      | -                            |
| Float   | 4                  | 00.0f   | -                            |
| Double  | 8                  |         | -1.8E+308<br>to<br>+1.8E+308 |

### 13. Mutable classes

1. Objects whose values can be changed after initialisation.

(They contain objects.)

(a) Program:

```
public class mutable {  
    private String s;  
    mutable (String s) {  
        this.s = s;  
    }  
}
```

```

public String getName() {
    return s;
}
public void setName (String courseName) {
    this.s = courseName;
}
public static void main (String [] args) {
    mutable obj = new mutable ("Information Technology");
    System.out.println (obj.getName());
    obj.setName ("Java Programming");
    System.out.println (obj.getName());
}
}

```

Output → Information technology  
Java Programming.

(b) Immutable classes.

Objects whose values cannot be changed after initialisation.  
(They contain objects)

Code;

```

public class immutable {
    private final String s;
    immutable (final String s) {
        this.s = s;
    }
    public final String getName () {
        return s;
    }
    public static void main (String [] args) {
        immutable obj = new immutable ("Java Programming");
        System.out.println (obj.getName());
    }
}

```

Output → Java Programming.



(c.) Situations where mutable classes are more preferable than immutable classes.

- Mutable fields can be changed after object creation while immutable fields cannot.
- Mutable classes generally provide a method to modify the field value while immutable classes don't have any method to modify the field value.
- Mutable classes have Getter and Setter methods while immutable classes have only Getter method.

13. Explain two possible ways of implementing polymorphism, and show code.

Compile time polymorphism (Method overloading)

- Also known as static polymorphism. It is achieved by function or operator overloading. Occurs when we define multiple methods with different signatures.

Code:

```
class Shapes {  
    public void area () {  
        System.out.println ("Area");  
    }  
    public void area (int r) {  
        System.out.println ("Circle area =  $3.14 * r * r$ ");  
    }  
    public void area (double b, double h) {  
        System.out.println ("Triangle area =  $0.5 * b * h$ ");  
    }  
    public void area (int l, int b) {  
        System.out.println ("Rectangle area =  $l * b$ ");  
    }  
}  
  
class main {  
    public static void main (String [] args) {  
    }  
}
```



```

Shapes myShape = new Shapes ();
// Create a shapes object

```

```

myShape.area();
myShape.area(5);
myShape.area(6.0, 1.2);
myShape.area(6.2);
}
}

```

Output → Find area  
 Circle area = 78.5  
 Triangle area = 360  
 Rectangle area = 12.

• Run-Time polymorphism (Method overriding)

- Also known as Dynamic Method Dispatch. Process in which a function calls to the overridden method is resolved at Runtime. It is achieved by Method Overriding.

Code ↓  
 Java

```

class Java {
  // defining a method
  void run ()

```

```

{
  System.out.println ("Code is written");
}

```

// Creating a child class

```

class Java2 extends Java

```

// defining method in same way as ~~first one~~ parent class

```

void run ()

```

```

{
  System.out.println ("Code is running properly");
}

```

```

public static void main (String[] args) {

```

```

  Java2 obj = new

```

```

  Java2 (); // Creating object

```

```

  obj.run ();

```

```

  // Calling method

```

```

}

```

```

}

```

Output → Code is running properly.

13 String buffer class explanation:

2. This is a class that is used to create mutable (modifiable) (a) String objects. It is the same as String class except that it is mutable.

Methods:

- append (String s) - Used to append the specified string with this string.
- insert (int offset, String s) - Used to insert the specified string with this string at the specified position.
- delete (int startIndex, int endIndex) - Used to delete the string from specified startIndex and endIndex.
- reverse () - Used to reverse the string.
- capacity () - Used to return the current capacity.
- ensureCapacity (int minimumCapacity) - Used to ensure the capacity at least equal to the given minimum.
- charAt (int index) - Used to return the character at the specified position.
- length () - Used to return the length of the string.
- substring (int beginIndex) - Used to return the substring from the specified beginIndex.
- substring (int beginIndex, int endIndex) - Used to return the substring from the specified beginIndex and endIndex.

Syntax

```
StringBuffer ob = new StringBuffer (String);
```