6 marks

1) **Basic data types in java**

   DATA TYPES:

   - **Every variable in java has a data type.**
   - **Data types specify the size and type of values that can be stored.**
   - **Java language is rich in its data types.**
   - **Primitive types are also called intrinsic or built-in types.**
   - **Derived data types are also known as reference types.**
   - **Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.**

     There are two data types available in Java −

     - **Primitive Data Types**
     - **Reference/Non-Primitive Types**

Primitive Data Types

- **There are eight primitive datatypes supported by Java.**
- **Primitive data types are predefined by the language and named by a keyword.**
- **Let us now look into the eight primitive data types in detail**

byte

- **Byte data type is an 8-bit signed two's complement integer**
- **Minimum value is -128 (-2^7)**
- **Maximum value is 127 (inclusive)(2^7 -1)**
- **Default value is 0**
- **Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.**

- **Example: byte a = 100, byte b = -50**

short

- **Short data type is a 16-bit signed two's complement integer**
- **Minimum value is -32,768 (-2^15)**
- **Maximum value is 32,767 (inclusive) (2^15 -1)**
- **Short data type can also be used to save memory as byte data type.**
- **A short is 2 times smaller than an integer**
- **Default value is 0.**
- **Example: short s = 10000, short r = -20000**

int

- **Int data type is a 32-bit signed two's complement integer.**
- **Minimum value is - 2,147,483,648 (-2^31)**
- **Maximum value is 2,147,483,647(inclusive) (2^31 -1)**
- **Integer is generally used as the default data type for integral values unless there is a concern about memory.**
- **The default value is 0**
- **Example: int a = 100000, int b = -200000**

Long

- **Long data type is a 64-bit signed two's complement integer**
- **Minimum value is -9,223,372,036,854,775,808(-2^63)**
- **Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)**
- **This type is used when a wider range than int is needed**
- **Default value is 0L**
- **Example: long a = 100000L, long b = -200000L**

Float

- **Float data type is a single-precision 32-bit IEEE 754 floating point**
- **Float is mainly used to save memory in large arrays of floating point numbers**
- **Default value is 0.0f**
- **Float data type is never used for precise values such as currency**
- **Example: float f1 = 234.5f**

Double

- **double data type is a double-precision 64-bit IEEE 754 floating point**
- **This data type is generally used as the default data type for decimal values, generally the**
- **default choice**
- **Double data type should never be used for precise values such as currency**
- **Default value is 0.0d**
- **Example: double d1 = 123.4**

Char

- **Char data type is a single 16-bit Unicode character**
- **Minimum value is '\u0000' (or 0)**
- **Maximum value is '\uffff' (or 65,535 inclusive)**
- **Char data type is used to store any character**
- **Example: char letterA = 'A'**

Reference Data Types

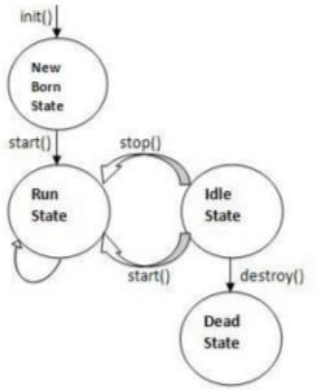
- **Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed.**
- **For example, Employee, Puppy, etc.**
- **Class objects and various type of array variables come under reference datatype.**
- **Default value of any reference variable is null.**
- **A reference variable can be used to refer any object of the declared type or any compatible**
- **type.**
- **Example: Animal animal = new Animal("giraffe");**

2) Applet Life cycle with diagram

LIFE CYCLE OF APPLET:

- Applet runs in the browser and its lifecycle-methods are called by JVM at its birth, its death and when it is running or idle.
- When an applet is made to work, it undergoes a series of changes as discussed below.
- Every Applet can be said to be in any of the following states
    1. New Born state
    2. Running state
    3. Idle state
    4. Dead or Destroyed state

The following figure represents the life cycle of the Applet



New Born State

- An applet enters into initialization state when the applet is first loaded into the browser by calling the init() method.
- The init() method is called only one time in the life cycle on an applet.
- The init() method retrieves the parameters through the PARAM tag of html file.
- At this stage, we may create Objects needed by applets, setup initial values, load images or fonts and setup colors.
- After the initialization of the init() method user can interact with the Applet.
- We can override this method.

Syntax:

```
public void init()
{
    Statements
}
```

Running State:

- After initialization, this state will automatically occur by invoking the start method of applet.
- The running state can be achieved from idle state when the applet is reloaded.
- This method may be called multiple times when the Applet needs to be started or restarted.
- If the user leaves the applet and returns back, the applet may restart its running.
- We can override this method.

Syntax:

```
public void start()
{
    Statements
}
```

Idle State:

- The idle state will make the execution of the applet to be halted temporarily.
- Applet moves to this state when the currently executed applet is minimized or when the user switches over to another page. At this point the stop method is invoked.
- From the idle state the applet can move to the running state.
- The stop() method can be called multiple times in the life cycle of applet.
- We can override this method.

Syntax:

```
public void stop()
{
    Statements
}
```

Dead State:

- When the applet programs terminate, the destroy function is invoked which brings an applet to its dead state.
- The destroy() method is called only one time in the life cycle of Applet like init() method.
- In this state, the applet is removed from memory.
- We can override this to cleanup resources.

Syntax

```
public void destroy()
{
    Statements
}
```

Display State:

- The applet is said to be in display state when the *paint* or *update* method is called.
- This method can be used when we want to display output in the screen.

- This method can be called any number of times.
- Overriding paint() method is a must when we want to draw something on the applet window.
- paint() method takes *Graphics* object as argument
- paint() method is automatically called each time the applet window is redrawn i.e. when it is maximized from minimized state or resized or when other windows uncover overlapped portions.
- It can also be invoked by calling "repaint()" method.

Syntax:

```
public void paint(Graphics g)
{
    Statements
}
```

3) Explain constructor with example program.

CONSTRUCTOR:

- **Constructor in java** is a *special type of method* that is used to initialize the object.
- Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.
- Constructs construct the values for object.

Rules for creating java constructor

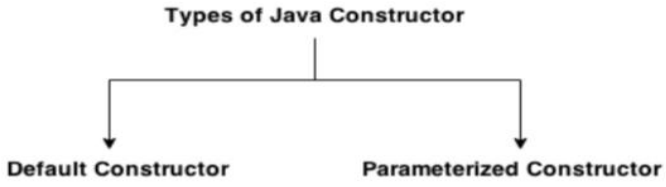
There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:

1. Default constructor (no-argument constructor)
2. Parameterized constructor



1. Java Default Constructor

A constructor that have no parameter is called default constructor.

Syntax:

```
<class_name>()
{
Constructor body
}
```

Example 1: (Default constructor)

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1
{
Bike1()    // constructor declared
{
System.out.println("Bike is created");
}
public static void main(String args[])
{
Bike1 b=new Bike1();  // constructor called
}}
```

Output:
Bike is created

Example 2: **(Default constructor)**

```
class Student
{
int id;
String name;
void display()
{
System.out.println(id+" "+name);
}
public static void main(String args[])
{
Student s1=new Student();          // Default constructor called
Student s2=new Student();          // Default constructor called
s1.display();
s2.display();
}}
```

Output:

0 null
0 null

In the above class, we are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

2. Java parameterized constructor

- A constructor that have parameters is known as parameterized constructor.
- Parameterized constructor is used to provide different values to the distinct objects.

Syntax:

```
<class_name>(parameter list)
{
Constructor body
}
```

Example: (parameterized constructor)

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student
{
  int id;
  String name;
  Student(int i, String n)
{
  id = i;
  name = n;
  }
  void display()
{
System.out.println(id+" "+name);
}
  public static void main(String args[])
{
  Student s1 = new Student(111,"Karan");
  Student s2 = new Student(222,"Aryan");
  s1.display();
  s2.display();
 }
}
```

Output:

```
111 Karan
222 Aryan
```

4) Explain how to define a class with example.

Defining a Class in Java:

Java provides a reserved keyword class to define a class. The keyword must be followed by the class name. Inside the class, we declare methods and variables.

In general, class declaration includes the following in the order as it appears:

- **Modifiers: A class can be public or has default access.**
- **class keyword: The class keyword is used to create a class.**
- **Class name: The name must begin with an initial letter (capitalized by convention).**
- **Superclass (if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.**
- **Interfaces (if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.**
- **Body: The class body surrounded by braces, { }.**

Syntax:

```
<access specifier> class class_name
{
// member variables
// class methods
}
```

Example :

```
// class definition
public class Calculate {

  // instance variables
  int a;
  int b;

  // constructor to instantiate
  public Calculate (int x, int y) {
    this.a = x;
    this.b = y;
  }

  // method to add numbers
  public int add () {
    int res = a + b;
    return res;
  }

  // method to subtract numbers
  public int subtract () {
    int res = a - b;
```

```
    return res;
}

// method to multiply numbers
public int multiply () {
    int res = a * b;
    return res;
}

// method to divide numbers
public int divide () {
    int res = a / b;
    return res;
}


// main method
public static void main(String[] args) {
    // creating object of Class
    Calculate c1 = new Calculate(45, 4);

    // calling the methods of Calculate class
    System.out.println("Addition is :" + c1.add());
    System.out.println("Subtraction is :" + c1.subtract());
    System.out.println("Multiplication is :" + c1.multiply());
    System.out.println("Division is :" + c1.divide());


}
```

Output :

```
C:\Users\Anurati\Desktop\abcDemo>javac Calculate.java

C:\Users\Anurati\Desktop\abcDemo>java Calculate
Addition is :49
Subtraction is :41
Multiplication is :180
Division is :11
```

5) Program on exception handling

12. ILLUSTRATION OF EXCEPTION HANDLING

```
class exceptiontest{
public static void main(String args[])
{
int d,a;
try{
d=0;
a=42/d;
System.out.println("This will not be printed");
}catch(ArithmeticException e)
{
System.out.println("Division by Zero");
}
System.out.println("After catch statement");
}
}
```

OUTPUT

```
D:\rr>javac exceptiontest.java

D:\rr>java exceptiontest
Division by Zero
After catch statement
```

6) Operators and its types

OPERATORS:

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

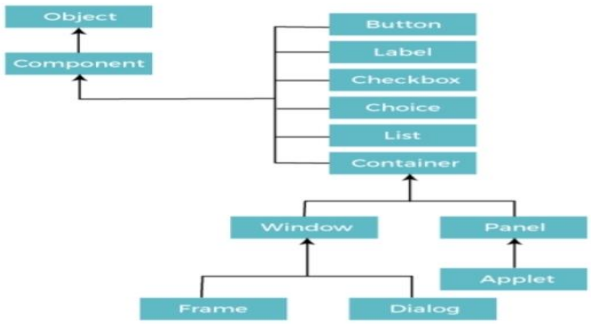
Operator Type	Category	Precedence
Unary	Postfix	*expr++ expr--*
	Prefix	*++expr --expr +expr -expr* ~ !
Arithmetic	Multiplicative	* / %
	Additive	+ -
Shift	Shift	<< >> >>>
Relational	Comparison	< > <= >= instanceof
	Equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	Ternary	? :
Assignment	Assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

7) Usage of any two AWT controls.

Java AWT Classes

- **Component**
- **Container**
- **Window**
- **Panel**
- **Frame**

The hierarchy of Java AWT classes are given below.



Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

Container

The Container is a component in AWT that can contain another components like buttons textfields, labels etc. The classes that extends Container class are known as container such as **Frame, Dialog** and **Panel**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

Types of containers:

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

Panel

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

Frame

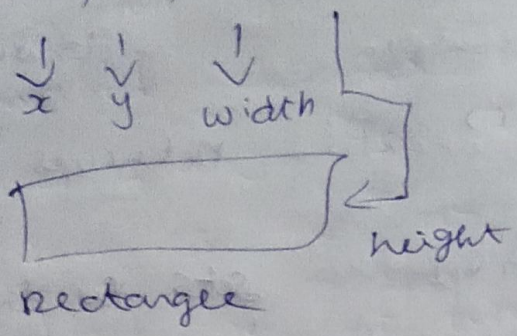
The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

8) How to draw lines and rectangles.

~~Line~~ Drawing Lines and Rectangles

* The simplest shape used in graphics class is line.
* The drawLine() Takes two pairs of coordinates (x_1, y_1) and (x_2, y_2) as arguments and draws a line between them.
* Example : g.drawLine(10, 10, 50, 50);
* The drawRect() method used to draw a rectangle. This method takes four arguments. The first two arguments represents x and y coordinates of the Top corner of the rectangle and remaining two arguments represents the width and height of the rectangle.

g.drawRect (10, 60, 40, 30);

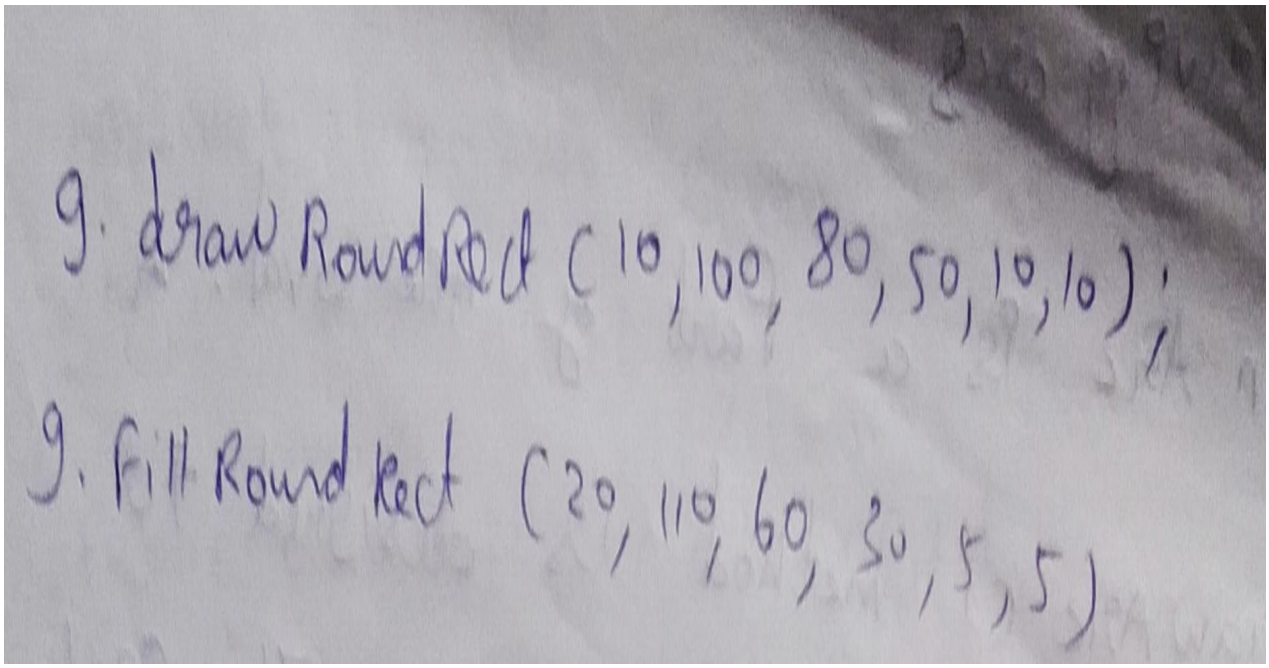


Rectangle

* To draw a solid box use fill rect method. This also takes 4 parameters corresponding to the starting point width and the height of the rectangle.

* g.fillRect (60, 10, 30, 50);

To draw rounded rectangles use drawRoundRect() and fillRoundRect() methods. This method use extra two arguments representing the width and height of the angle of the corners.



9) Border layout with example

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window.

The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class: Using BorderLayout() constructor

```
import java.awt.*;
import java.applet.*;
 class BorderLayoutExample extends Frame
{ BorderLayoutExample()
{
 setLayout(new BorderLayout());
add(new Button("NORTH"),BorderLayout.NORTH);
add(new Button("SOUTH"),BorderLayout.SOUTH);
add(new Button("EAST"),BorderLayout.EAST);
 add(new Button("WEST"),BorderLayout.WEST);
```

```
 add(new Button("CENTER"),BorderLayout.CENTER);
}
}
class BorderLayoutJavaExample
{
public static void main(String args[])
{
BorderLayoutExample frame = new BorderLayoutExample();
frame.setTitle("BorderLayout in Java Example");
frame.setSize(400,150);
frame.setVisible(true);
```

Output:

20.2Java Try Catch