Java

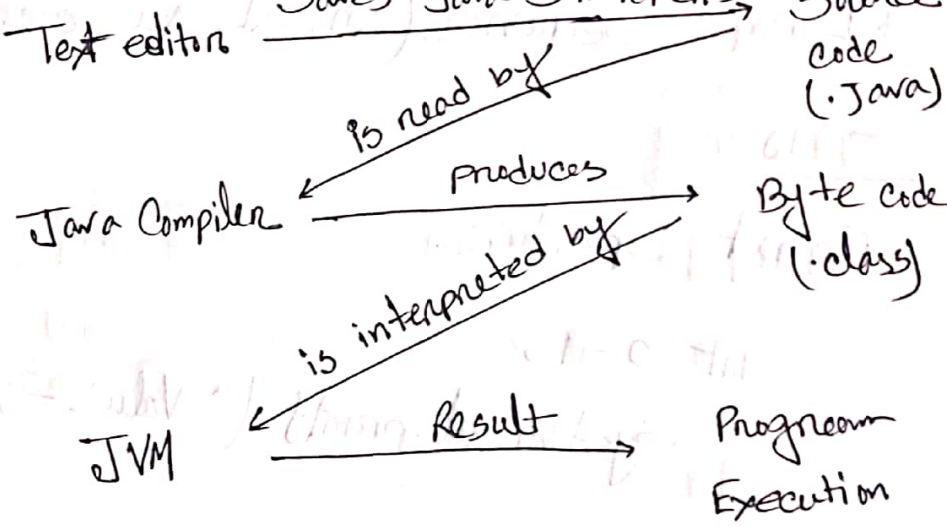
T-1

* Originally developed by Sun microsystem. (1995)
  Now owner is Oracle.

* Father → James Gosling

Execution Step:

1. Edit
2. Compile
3. Load
4. Verify
5. Execute



* It is high level programming Language.

Feature:

① Platform Independent
② Object oriented
③ Support web based application
④ Robust
⑤ Secure
⑥ Multi-threaded

```
class Hello_world{
    public static void main(String[] args){
    System.out.println("Hello World!");
    }
}
```

* Escape sequence (\n, \t, \a etc.)

T-10

Format specifier:

```
int b=10;
System.out.printf("Value = %d", b);
```

Taking Input:

```
import java.util.Scanner;

public class TakingInput{
    public static void main(String[] args){
        int num;
        Scanner input = new Scanner(System.in);
        num = input.nextInt();
        System.out.println("Number is = "+num);
    }
}
```

The ?: notation

```
z = (x<y) ? x : y;
```

switch

```
switch (a) {
    case '0':
        System.out.println ("zero");
        break;
    case '1':
        System.out.println ("one");
        break;
    default:
        System.out.println ("Two+");
        break;
}
```

Array

```
int [] number; // Declaration
number = new int[10]; // creation
```

Or,

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

```
int len = number.length; // Size of Array.
```

For each

```
String [] names = {" Anis", "Rayhan", "Mim", "Dim"};

for (String x : names) {
    System.out.println(x);
}                    // Same as Normal Loop.
```

2D Array

```
int[][] number = new int[3][4];
                      row  col
```

ArrayList

```
import java.util.ArrayList;

public class Demo {
    public static void main(String[] args) {
        ArrayList<Integer> number = new ArrayList<Integer>();
        System.out.println(" Size = " + number.size());

        number.add(10);
        number.add(2,13);
        System.out.println(number);
    }
}
```

Or,

```
for(int x: numbers){
    System.out.println(x);
}

Collections.sort(numbers);
```

StringBuffer (One kind of Same StringBuilders)

```
StringBuffer sb = new StringBuffer("Rayhan");
```

// Same of String, you can change or edit it without any other string.

Wrapper Class (.valueOf())

Autoboxing = converting primitive to object

Unboxing = converting object to primitive

Primitive	object
int	Integer
float	Float
char	Characters
--- etc.	--- etc.

Conversion between String and Primitive Data type:

```
int i = 100;
String s = Integer.toString(i);
System.out.println("s is :" + s);
```

or,

```
String s = "235789";
Integer I = Integer.parseInt(s); // or. .valueof()
System.out.println("I is :" + I);
```

Binary, Octal, Hexa → Decimal

```
String Binary = "10101011D";
Integer Decimal = Integer.parseInt(binary, 2);
System.out.println(Decimal);
```

Vice-Versa

```
int decimal = 15;
String binary = Integer.toBinaryString(decimal);
System.out.println(binary);
```

Mid Exam

byte = 8 bits
float = 32 "
Short = 16 "
double = 64 "
int = 32 "
long = 64 "

Method Overloading

Same name but parameter আলাদা

Method Overriding.

Name and parameter same.

Encapsulation

public = access can all
protected = access can only own project or file.
private = none can access (but using set, get function access)

1. Inheritance and Polymorphism
2. Exception Handling
3. Java Swing
4. Final and Abstract class
5. I/O Statement

Exception:

An exception is an abnormal condition that arises in a code sequence at run time.

* Preventing overriding using **final**

Java Environment

The development tools are part of the system known as Java Development Kit (JDK).

The classes and methods are part of the Java Standered Library (JSL), also known as the Application Programming Interface (API).

Java Development Kit [used for developing and running Java Programs]

1. appletviewer (for viewing Java applets)
2. javac (Java compiler)
3. java (Java Interpreter)
4. javap (Java disassembler)
5. javah (for C header files)
6. javadoc (for creating HTML documents)
7. jdb (Java debuggers)

Java Runtime Environment

JRE facilities the execution of programs developed in Java. It primarily comprises the following:

1. Java Virtual Machine (JVM): It is a program that interprets the intermediate Java byte code and generates the desired output. It is because of byte code and JVM concepts that programs written in Java are highly portable.

2. Runtime Class Libraries: These are a set of core class libraries that are required for the execution of Java programs.

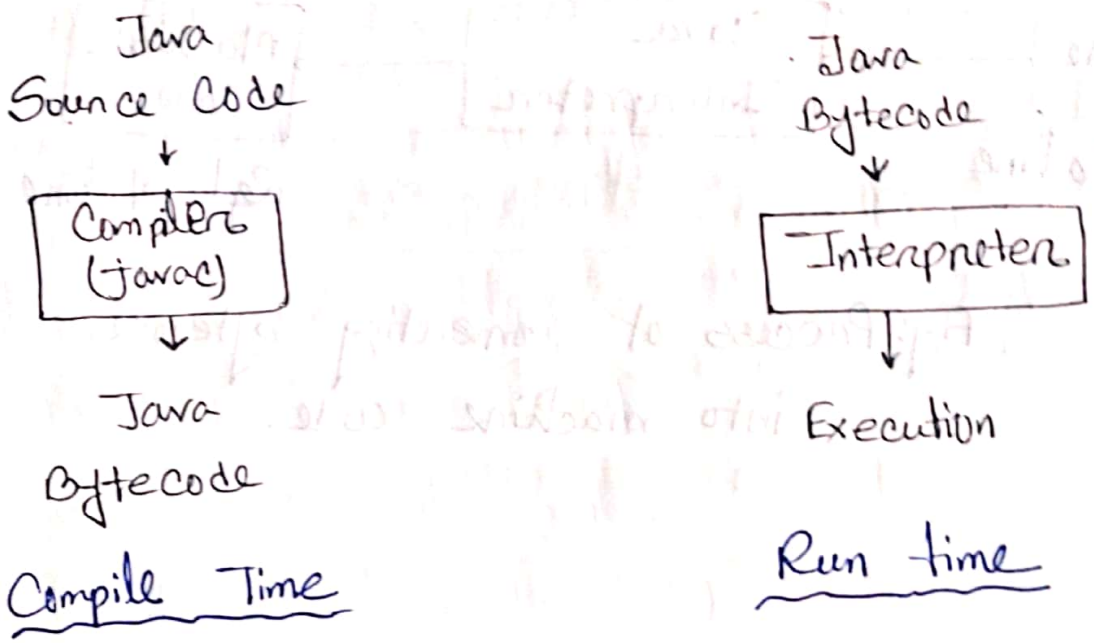
3. User Interface toolkits: AWT and Swing are examples of toolkits that support varied input methods for the users to interact with the application program.

4. Deployment technologies: JRE comprises the following key deployment technologies:

   @ Java plug-in: Enables the execution of a Java applet on the browser.

(b) Java Web Start: Enables remote deployment of an application. With Web start, users can launch an application directly from the Web browser without going through the installation procedure.

Why Java is Cross-platform?



Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the size of the primitive data types are machine independent.

Java Virtual Machine (JVM)

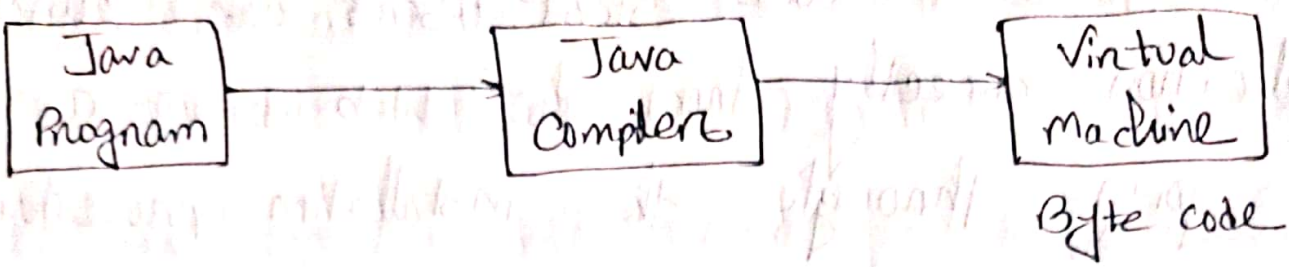


Fig: Process of compilation.

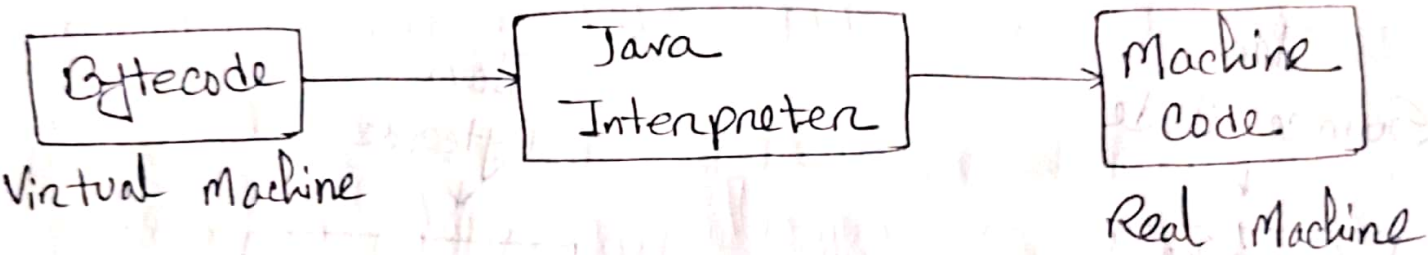
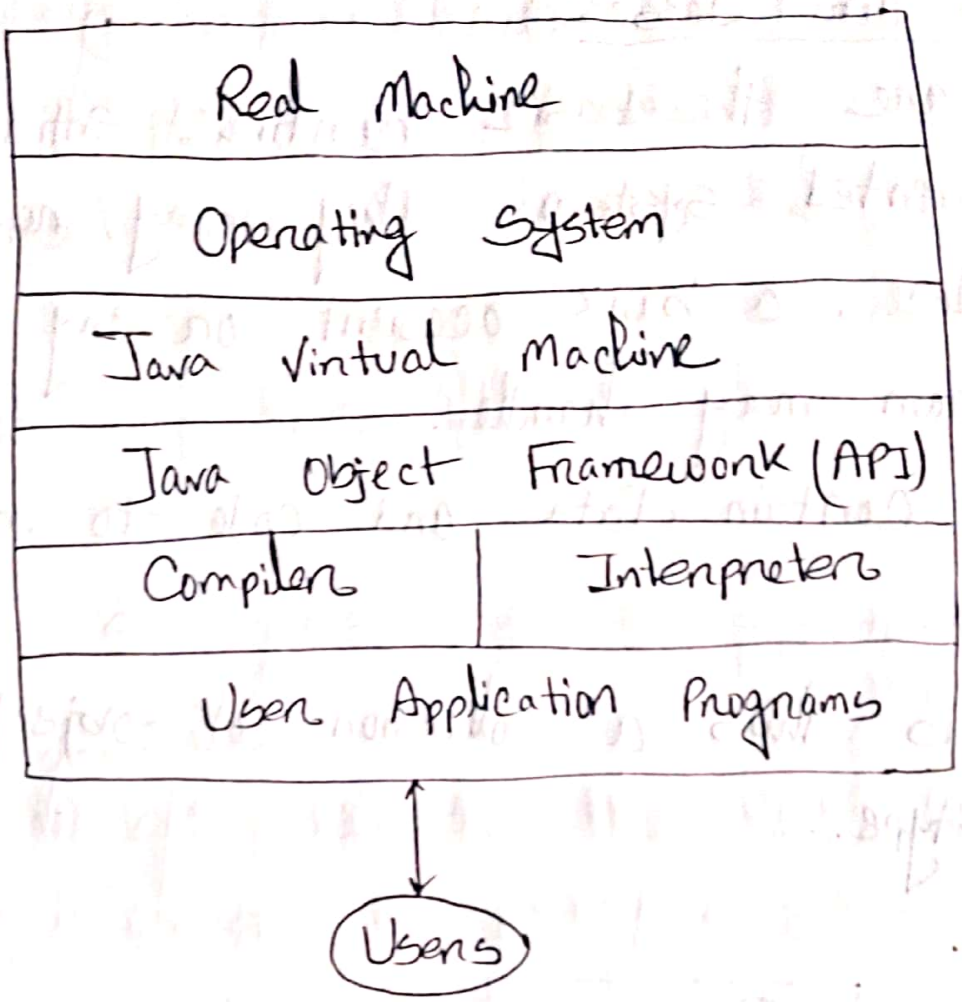


Fig: Process of converting bytecode into machine code.

Layers of interactions for Java programs.



Basic of OOP

① Objects and Classes:

Objects are the basic runtime entities in an object-oriented system. They may represent a person, a place, a bank account or any item that the program may handle.

Objects contain data and code to manipulate that data.

A class is thus a collection of objects of similar type.

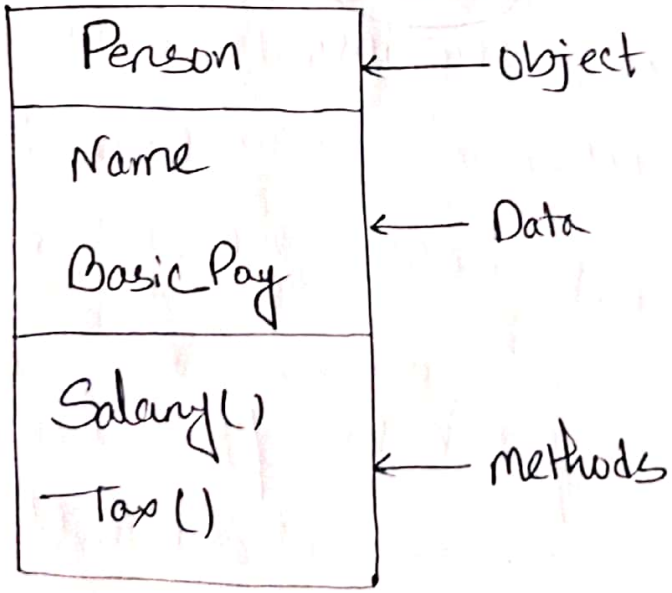


Fig: Representation of an object.

② Data Abstraction and Encapsulation:

The wrapping up of data and methods into a single unit (called class) is known as Encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it. This insulation of the data from direct access by the program is called data hiding. Encapsulation makes it possible for objects to be treated like 'black boxes' each performing a specific task without any concern for internal implementation.

Fig: Encapsulation - objects as "black boxes".

Abstraction refers to the act of representing essential features without including the background details or explanations.

③ Inheritance:

Inheritance is the process by which objects of one class acquire the properties of objects of another class. Inheritance supports the concept of hierarchical classification. In OOP, the concept of inheritance provides the idea of reusability.

A class created by extending another class is called a subclass.

The class used for the basis is called the Superclass.

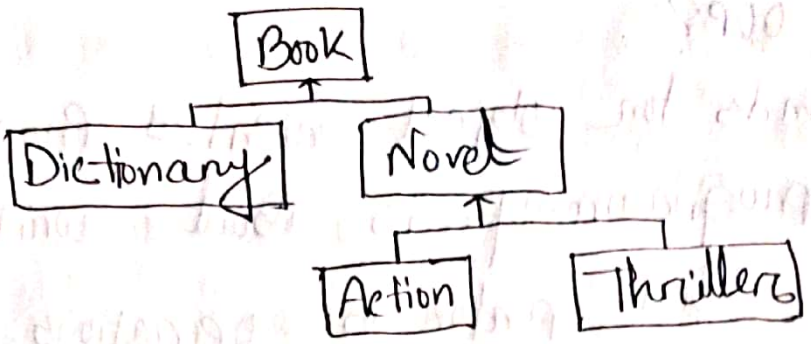


Fig: Inheritance

④ Polymorphism:

Polymorphism means the ability to take more than one form. Polymorphism is extensively used in implementing inheritance.

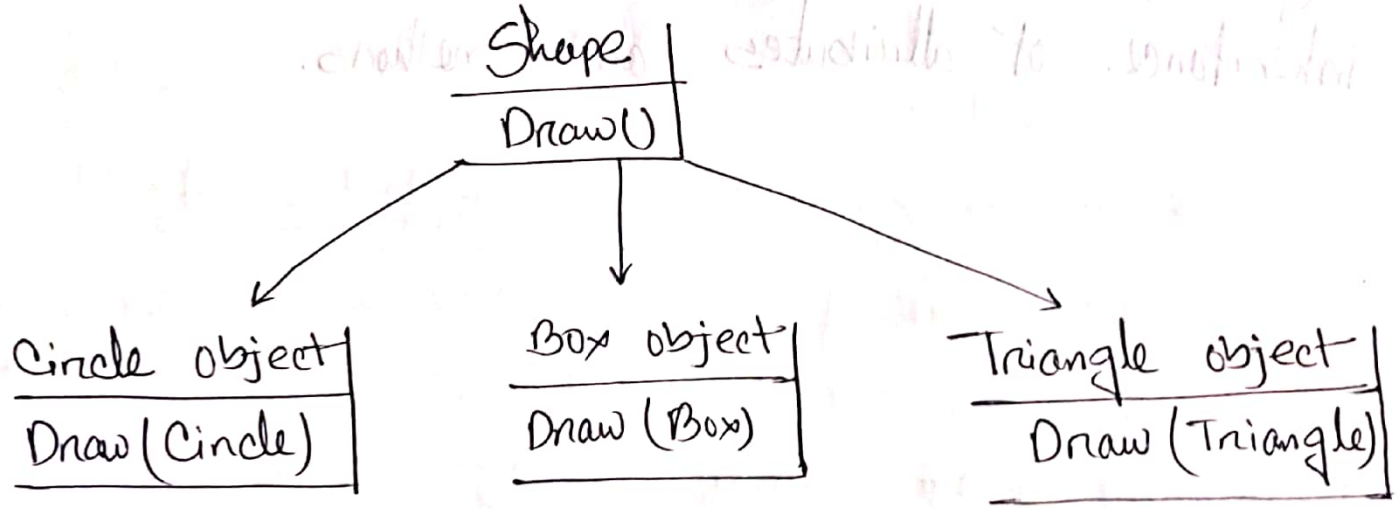


Fig: Polymorphism.

What is OOP?

Ans: OOP stands for object-oriented Programming. Procedural programming is about writing procedures or methods that performs operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

or,

OOP is a style of programming characterized by the identification of classes of objects closely linked with the methods (functions) with which they are associated. It also includes ideas of inheritance of attributes and methods.

Binding

Binding is a mechanism creating link between method call and method actual implementation. As per the polymorphism concept in java, objects can have many different forms. objects forms can be resolved at compile time and run time.

If linking between method call and method implementation is resolved at compile time then we call it static binding.

Or, it is resolved at run time then it dynamic binding. Dynamic binding uses objects to resolve binding but static binding use type of the class and fields.

Static Binding	Dynamic Binding
① It is resolved at comiple time.	It is resolved at run time.
② Static binding use type of the class and fields.	Dynamic binding uses object to resolve binding.
③ Overloading is an ex.	method overriding is an ex.
④ Privates, final and static methods and variables uses static binding.	Virtual methods are used in Dynamic binding.

Ex:

```
public class FastFood {
    public void create(){
    System.out.println("Creating in Fast-Food class");
    }
}

public class Pizza extends FastFood {
    public void create(){
    System.out.println("Creating in Pizza class");
    }
}

public class Main {
    public static void main(String[] args){
    // Static Binding
    FastFood f = new FastFood();
    f.create();
    // Dynamic
    FastFood p = new Pizza();
    p.create();   → Creating in Pizza class.
    }
}
```

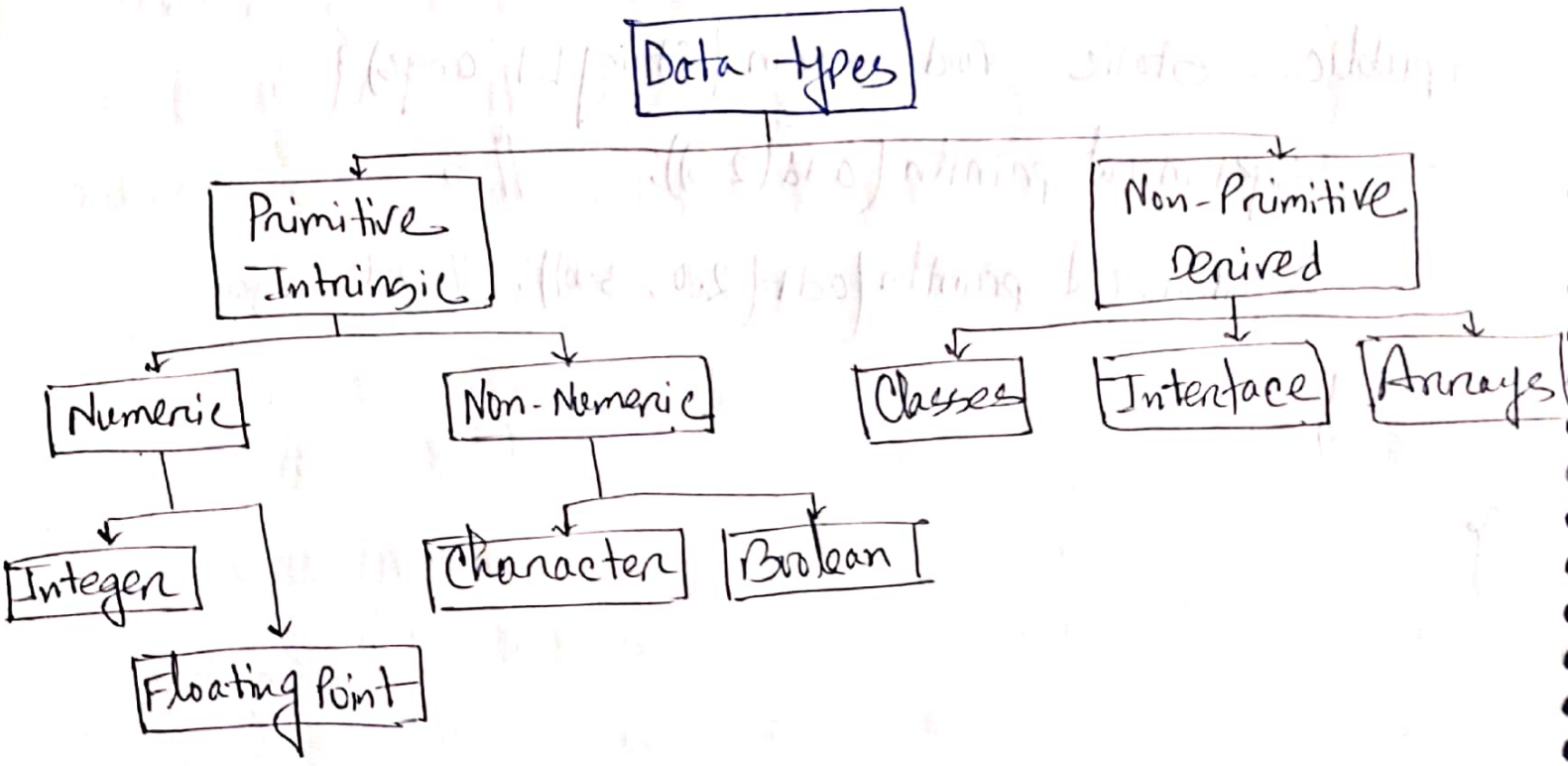
Static Binding

```
public class Rayhan {
    public static int add(int a, int b){
        return a+b;
    }

public static double add(double a, double b){
        return a+b;
    }
}

public static void main(String[] args){
    System.out.println(add(2,3));      // 5
    System.out.println(add(2.0, 3.0)); // 5.0
    }
}
```

Java Program Structure

1. Documentation Section
2. Package Statement
3. Import "
4. Interface "
5. Class Definitions
6. Main method class

```
{
  Main method definition
}
```



Primitive Types

byte → 8 bit
short → 16 bit
int → 32 "
long → 64 "
} Integer

float → 32 "
double → 64 "
} Floating point

char → 16 "
boolean → either true or false
} other types

* int a = 30; // initialization
* long b;
  b = -20; // assignment

* '0' = 48
  '1' = 49
  'A' = 65
  'a' = 97

Scope of Variables

① instance variables
② class variables
③ local variables

$a \% b = a - (a/b) * b$

* When a program breaks the sequential flow and jumps to another part of the code, it is called branching. (If, Switch, Conditional operators)

Switch

Ex:
```
switch(ch) {
    case 'a': case 'A':
    case 'e': case 'E':
    . . . . . .
    System.out.println("vowel");
    break;
    default:
    System.out.println("not a vowel");
    break;
}
```

- A class is a user-defined data type with a template that serves to define its properties.

```
class Rectangle {
    int length, width;
    void getData(int x, int y) {
        length = x;
        width = y;
    }
}
```

- An object in Java is essentially a block of memory that contains space to store all the instance variables. Creating an object is also referred to as instantiating an object.

```
Rectangle rect1 = new Rectangle();
```

→ Default constructor.

Action	Statement	Result
Declare	Rectangle Rect1;	null rect1
Instantiate	rect1 = new Rectangle();	rect1

Rect1 is a reference to Rectangle objects

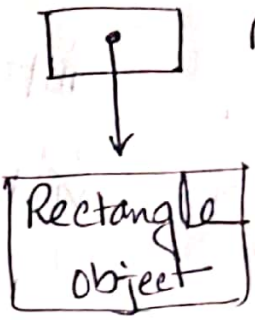


Fig: Creating object references.

- Java supports a special type of method, called a constructor, that enables an objects to initialize itself when it is created.

```
Rectangle () {
    length = 0;
    width = 0;
}
```

Default constructor

```
Rectangle (int x, int y) {
    length = x;
    width = y;
}
```

Parameterized Constructor

In Java, it is possible to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading.

Method overloading is used when objects are required to perform similar tasks but using different input parameters.

This process is known as Polymorphism.

Static

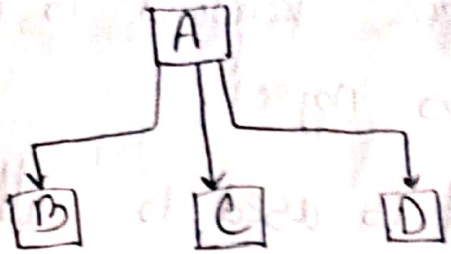
```
class Math{
    static float mul(float x, float y){
        return x*y;
    }
    static float div(float x, float y){
        return x/y;
    }
}

class Main{
    public static void main(String[] args){
        float a = Math.mul(4.0, 5.0);
        float b = Math.div(a, 2.0);
        cout << a << b;
    }
}
```

Limitations:

1. They can only call other static methods.
2. " " " access static data.
3. They can not refer to this or super in any way.

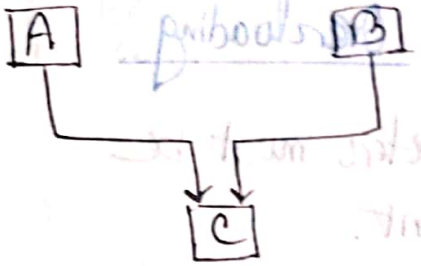
Inheritance

Single Inheritance



Hierarchical inheritance

multilevel inheritance



Mutiple inheritance
(Java does not support)

The mechanism of deriving a new class from an old one is called inheritance.

iii) Super

"Super" keyword is used to refer immediate super class object.

① It is used to call super class instance variable.
② " " " " " " " method.
③ " " " " " " " constructors.

Overloading	Overriding
1. Parameter must be different.	Parameter must be same.
2. It occurs within the same class.	It occurs between two classes -sub and super class.
3. Inheritance is not involved.	Inheritance is involved.
4. Return type may or may not be same.	Return type must be same.
5. One method does not hide another.	Child method hides parent another.

* We can prevent overriding using "final".
* A class that cannot be subclassed is called a final class.

Ex:

```
public abstract class Shape {
    double dim1, dim2;
    Shape (double dim1, double dim2){
        this.dim1 = dim1;
        this.dim2 = dim2;
    }
    abstract void area();
}

public class Rectangle extends Shape{
    Rectangle (double dim1, double dim2){
        super (dim1, dim2);
    }
    void area(){
    Cout << dim1 * dim2;
    }
}
```

And so on for Triangle, Circle etc....

```
public class Test {
    public static void main (String [] args) {
        Shape s;
        s = new Rectangle (20,30);
        s.area();

        s = new Triangle (10,30);
        s. area();

        ----

    }
}
```

Array

```
int num[];  // Declare
num = new int[10];  // Creating memory
```

or, `int num[] = new int[10];`

* `int num[][] = new int[5][7];`
* `int size = num.length;`
* `String name = new String("Rayhan");`

- StringBuffer is a peer class of String. While String creates strings of fixed-length, StringBuffer creates strings of flexible length that can be modified in terms of both length and content.

Wrapper class

Autoboxing = Converting primitive to object
Unboxing = " object to primitive

//Autoboxing

```
int x = 30;
Integer y = Integer.valueOf(x);
```

or, `Integer y = x;`

//Unboxing

```
Double dx = new Double(10.25);
double e = dx.doubleValue();
```

or, `double e = dx;`

Primitive to String.

```
double i = 100;
String s = Double.toString(i);
```

String to Primitive

```
String s = "100";
double i = Double.parseDouble(s);
```

Interface

An interface is a collection of abstract methods.

```
interface xyz {
public static final int code = 1001;
public static final String s = "Fan";
  public abstract void display();
  }
```

Class	Interface
1. It can be instantiated by declaring objects.	It can not be instantiated. It can only be inherited by a class.
2. It may or may not contains Construction.	It doesn't Contain construction
3. All methods in class can or can not be abstract.	All the methods in interface are abstract.
4. Can have instance variables.	Can't have instance variables.
5. Classes can not extened multiple inheritance.	Interface can extended multiple inheritance.

6. The members of a class can be constant or variables.	The members of an interface are always declared as constant. That is their values are final.
7. It can various access modifier like public, private, protected.	It can only use the public access specifier.

Multithreading

It is a programming concept in which a programme or a process is devided into two or more subprograms or threads that are executed at the same time in parallel.

It supports execution of multiple parts of a single program simultaneously.

The processor has to switch between different parts or threads of a program.

It is highly efficient.

A thread is the smallest unit in multithreading.

It helps in developing efficient programs.

Multitasking

It is an operating system concept in which multiple tasks are performed simultaneously.

It supports execution of multiple programs simultaneously

The processor has to switch between different programs or processes.

It is less efficient in comparism to multithreading.

A program or process is the smallest unit in a multitasking environment.

It helps in developing efficient operating systems.

It is cost-effective in Case of context switching.	It is expensive in case of context switching.

Ex!

```
Class A extends Thread {
    public void run(){
        for(int i=1; i<=10; i++){
            cout<<"Count: "+i;
        }
    }
}

class B extends Thread {
    public void run(){
        for(int i=10; i<=20; i++){
            cout<<"Count: "+i;
        }
    }
}
```

```
class ThreadTest{
    public static void main(String[] args){
        A a = new A();
        a.start(); //or, a.run();

        B b = new B();
        b.start();

    or,
        new A().run(); --- and so on. ---
    }
}
```

Stopping a Thread

```
a.stop();
```

Blocking a Thread

```
sleep()     //blocked for a specified time
suspend()   //  "   until further orders
wait()      //  "     "  certain condition occurs
```

Life Cycle of a Thread

1. Newborn
2. Runnable
3. Running
4. Blocked
5. Dead

- Killing threads in Java
  - suspend()
  - resume()
  - stop()

An exception is an abnormal condition that arises in a code sequence at ruen time.

Categorized:

1. Checked exceptions : extended from java.lang.Exception
2. Unchecked " : extended " java.lang.RuentimeException

Checked

1. User mis-types input
2. Web page not available
3. File not found.
4. No such field
5. Interrupted
6. No such method
7. class Not found

Unchecked

1. Array index out of bound
2. Method called on a null object.
3. Divide by zero.
4. No such element
5. Undeclared Throwabl
6. Empty Stack
7. Arithmetic
8. Null pointer
9. Security

Sys errors

1. Out of memory
2. Bug in the actual language implementation.

* If an exception occurs within the [try] block, it is thrown.

→ Code within [catch] block catch the exception and handle it

→ System generated exceptions are automatically thrown by the Java run-time system.

→ To manually [throw] an exception, use the keyword throw

→ Any exception that is thrown out of a method must be specified as such by a [throws] clause

→ Any code that absolutely must be executed before a method returns is put in a [finally] block.

- An **Error** indicates a serious problem that a reasonable application should not try to catch.
  1. ClassFormatError
  2. InstantiationError
  3. InternalError
  4. NoSuchMethodError
  5. OutOfMemoryError
  6. StackOverflowError
  7. VirtualMachineError

- RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the JVM.
  1. NullPointerException
  2. ArrayIndexOutOfBoundsException
  3. NegativeArraySizeException
  4. ClassCastException
  5. NumberFormatException
  6. SecurityException

Ex.

```
import java.lang.Exception;
Class InvalidRadiusException extends Exception {
    private double r;
    public InvalidRadiusException(double radius){
        r = radius;
    }
    public void printError(){
        cout << "Radius " << r << " is not valid";
    }
}
Class cincle{
    double x, y, r;                    throws InvalidRadiusExcep
    public cincle(double x, double Y, double r){
                                       dius
        if(r<=0){
            throw new InvalidRadiusException(radius);
        }
        else{
            x=X; y=Y; r=radius;
        }
    }
}
```

```
class CircleTest{
  public static void main(String[] args){
    try{
      Circle c1 = new Circle(10, 10, -1);
      Cout<<"circle created";
    }
    catch(InvalidRadiusException e){
        e.printError();
    }
  }
}
```

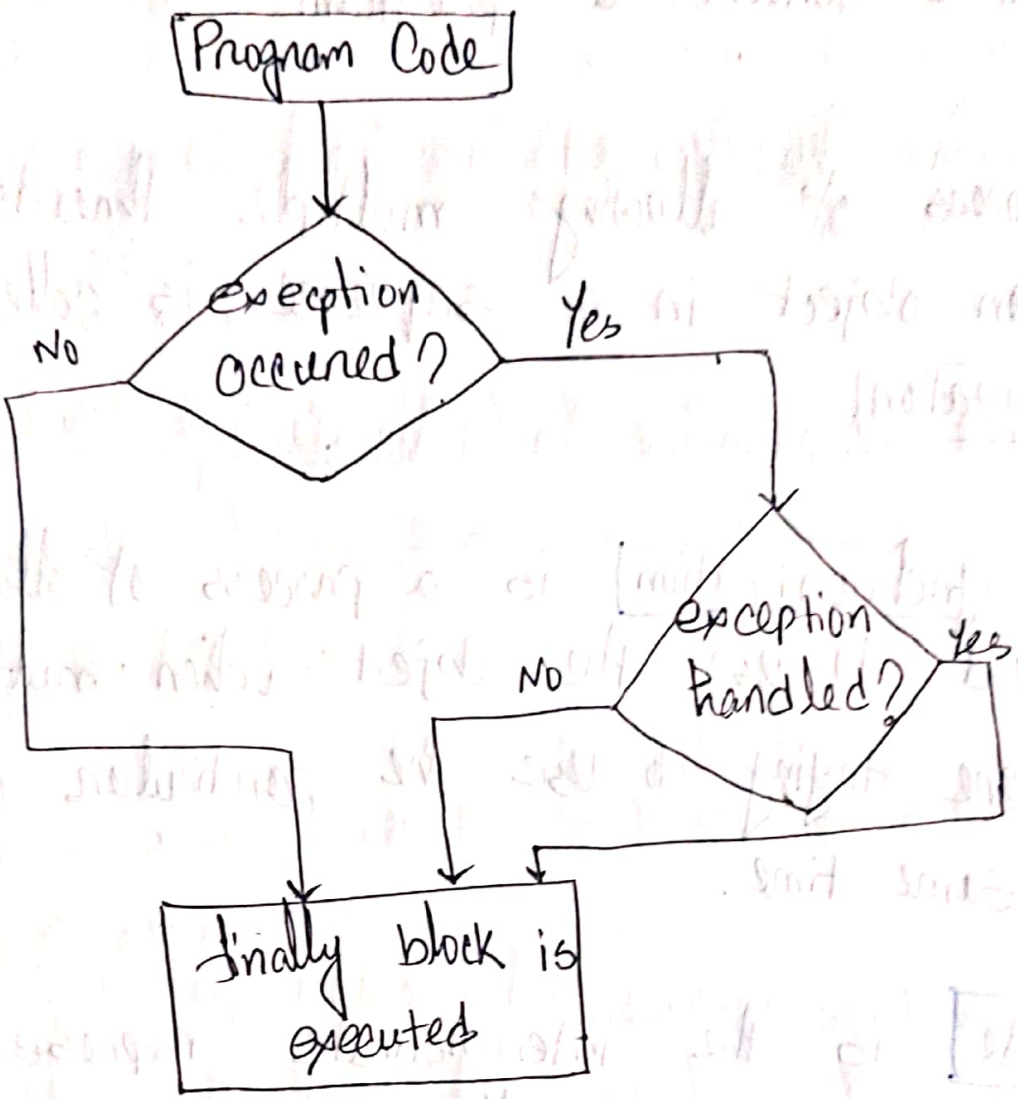
Create a file

```
import java.io.File;
Class Test{
  public static void main(String[] args){
      File f1 = new File("Student.txt");
      File f2 = new File("Teacher.txt");

    try{
        f1.createNewFile();
        f2.createNewFile();
    }
    catch(Exception e){
      cout<<e;
    }
  }
}
```

this	super
this keyword mainly represents the current instance of a class.	super keyword represents the current instance of a parent class.
this keyword used to call default constructor of the same class.	super keyword used to call default constructor of the parent class.
this keyword used to access methods of the current class as it has reference of current class.	One can access the method of parent class with the help of super keyword.
this keyword can be referred from static context that is can be invoked from static instance.	On the other hand super keyword can't be referred from static context that is can't be invoked from static instance.

throw	throws
Throw is a keyword which is used to throw an execption explicity in the program inside a function or inside a block of code.	Throws is a keyword used in the method signature used to declare an exception which might get thrown by the function while executing the code.
We can propagate only unchecked exception.	both checked and unchecked exceptions can be declared.
We cannot throw multiple exception with throw keyword.	We can declare multiple exception exception.
Syntax wise throw keyword is followed by the instance variable.	... class name.
used within the method.	used with the method signature.

Finally Block



* A **Thread** is a single sequential flow of control within a program.

* The process of allowing multiple threads to modify an object in a sequence is called **Synchronization**

Thread Synchronization is a process of allowing only thread to use the object when multiple threads are trying to use the particular object at the same time.

* **Bytecode** is the intermediate representation of a Java program, allowing a JVM to translate a program into machine level assembly instructions.

Checked	Unchecked
occurs at compile time.	occurs at reen time.
The compiler checks a checked exception.	does not
These types of exceptions can be handled at the time of compilation.	can not. - because they get generated by the mistakes in the program.
They are sub-class of the exception class.	They are reentime exceptions and hence are not a part of the Exception class.
The JVM needs the exception to catch and handle.	does not.

import java.lang.Math;

- JOptionPane.showMessageDialog(null, "How are you?");

* int choice = JOptionPane.showConfirmDialog(null, "Want you?");
  if( )
  - - -
  else - - -
  - - -

* String name = JOptionPane.showInputDialog(
  null, "what's your name?");