JAVA NOTES

ADIKA KARNATAKI

OOPP Java

```
public class FirstJavaProgram
    {
      public static void main (String[] args)
      {
       System.out.println (" First program in Java");
      }
    }
```

public - Modifier / Access specifier
public, private, protected, default

class - keyword (type of object)

classname - Identifier decided by programmer

static - without creation of object for a class, methods / variables are accessed outside the current class

void - method is returning empty data

main - Execution of program starts here

String[]args - Input taken in Java is String form input and saved in args array

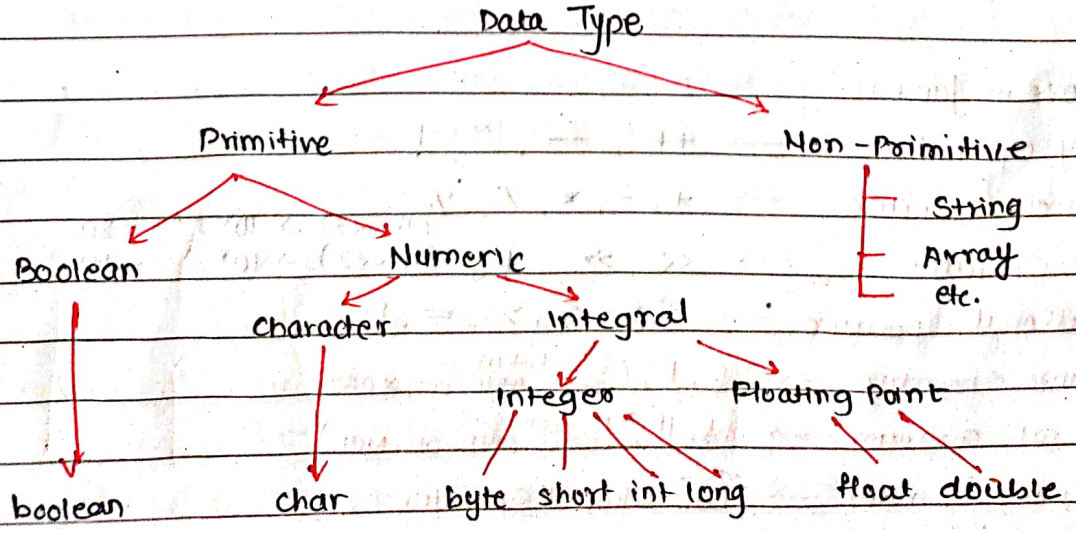
{...} - within opening and closing brackets, we write code

println - It is the java statement to display output

javac - Compiler of Java

java - Interpreter of Java

// Data types in Java



// Decision making in Java

1. if
2. if else
3. Nested if-else
4. if-else-if
5. switch-case
6. jump → break, continue, return

- If-else ladder
  - User can decide among multiple options
  - If statements are executed from top to down.
  - As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed.
  - If none of the condition is true, then the final else statement will be executed.
- Switch case
  - The switch statement is a multiway branch statement. It provides an easy way of execution of different parts of code based on the value of expression.
- Jump Statements

  Java supports three jump-statement : break, continue, return

  These statements transfer control to other part of program.

  break :- terminate sequence in switch
  to exit a loop
  used as civilized form of goto.

  Continue :- used to continue executing statements in a loop after any skip type (if) condition. It is used to skip and continue in a loop.

  return :- used to explicitly return from a method. It causes program control to transfer back to caller of method.

// Loops

while loop- entry level control structure

for loop

do while loop- checks for condition after executing statements

Exit control loop

// ARRAYS IN JAVA

- Java array is an object which contains elements of similar data type.
- The elements of an array are stored in a contiguous memory location.
- It is a data structure where we store similar elements.
- We can store only a fixed set of elements in a Java array.

int [] myarray = new int[3]; myarray[0]=2, myarray[1]=7,--

Indexes: The values can be retrieved from array using index.

The first value is found from index 0 and last from length-1.

Initializing array with {}

int [] myarray = {2,8,12}

For-each loop

for(datatype variable : Array)

e.g. for(int i : myList)

Two dimensional array:-

int [][] twoD-arr = new int [10][20];

// Command Line Arguments

- It is user input from the command line. Argument array is ~~partially~~ initialized automatically for you.

// Array Bounds Checking

- Whenever array is accessed, the index is checked to ensure that it is within the bounds of array.
- Attempts to access an array element outside the bounds of array will cause an ArrayIndexOutOfBounds exception to be thrown.

Command line e.g.:-

```
public class CommandLine {
    public static void main (String [] args) {
        System.out.println(args[0]);
        System.out.println(args[1]);
    }
}
```

// Strings in JAVA

- Every string we create is an object of type String.
  String constants are actually string objects.
- Objects of String are Immutable ie. once a string is created, its contents cannot be altered.

- String has been widely used as a parameter for many Java classes. e.g. for opening network connection we can pass hostname and portnumber as string. we can pass database URL as string for opening database connection
we can open any file in Java by passing name of file as argument to File I/O classes.
In case if String is not immutable, this would lead to serious security threat, means some one can access to any file for which he has authorization and then can change filename either deliberately or accidentally and gain access to file.

In Java, 4 predefined classes are provided that either represent Strings or provide functionality to manipulate them.

- String
- StringBuffer
- StringBuilder
- StringTokenizer

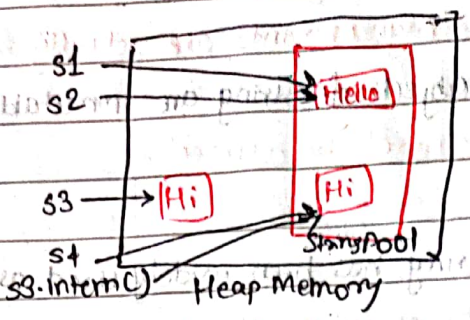
} javalang package
Implement CharSequence Interface.

String Handling
Perform operations on strings:
for e.g. Compare two strings, search for substring
concatenate two strings, change case of letters within string.

String class constructor
public String()
public String(String)
public String(char[])
public String(byte[])
public String(char[], int offset, int no-of-chars)

String pool.

```
public class JavaStringPool {
public static void main (String[]args){
String s1 = "Hello";
String s2 = "Hello";
String s3 = new String ("Hi");
String s4 = "Hi";
System.out.println (s1 == s2);
System.out.println (s3 == s4);
s3 = s3.intern();
System.out.println (s3 == s4);
}
}
```



When the first string s1 created, there is no string with value "Hello" in stringpool. So a string "Hello" is created in pool and its reference is assigned to s1. When s2 created, there is already a string with value "Hello" in pool. The existing string reference is given to s2. Whenever we create s3, it's created in heap area because new operator is used. When compared s1==s2 its true because both refer to same object. When s4 is created, a new string with value "Hi" is created in pool and reference assigned to s4. When compared s3==s4, its FALSE because both point to different string objects.

When we call intern() method on s3, it checks if any string is there in pool with same value. and reference assigned to s3. Now s3==s4 is true because both variables are referring to same objects.

String Concatenation :-

```
String age = "9";
String s = "He is" + age + "years old";
sysout(s);
```

} concat string with string

```
int age = 9;
String s = "He is" + age + "years old";
sysout(s);
```

} concat String with other data types

There are two methods to concatenate strings:

```
String s = "Hello";
String r = "World";
String s1 = s.concat(r);
String s2 = "Hello".concat("World");
```

} concat() method.

Second method: Using '+' operator

// Methods of String class:

STRING LENGTH:

length() returns length of the string i.e. no. of characters.

STRING COMPARISON:

- Using equals() method
- Using == Operator
- By CompareTo() method

```
String s' = "Hell";
String s1 = "Hello";
String s2 = "Hello";
String s3 = "Java";
// Using equals() method
sysout(s1.equals(s2)); // true
Sysout(s.equals(s1)); // false
// Using == operator
String s4 = new String("Java");
sysout(s1==s2); // true (because both refer to same instance)
Sysout(s3==s4); // false (because s4 refers to instance in nonpool)
// By compareTo() method
sysout(s.compareTo(s2)); // return -1 because s1<s2
sysout(s1.compareTo(s2)); // return 0 because s1==s2
sysout(s2.compareTo(s)); // return 1 because s2>s
```

// SUBSTRING:

A part of a String is called Substring. It is subset of another String.
We can get substring from given String object by one of these methods.

- public String substring(int startIndex);
- public String substring(int startIndex, int endIndex);

// java.lang.String class provides a lot of methods to work on string.

1. charAt() ⇒ returns char value at the given index number.
   Index no. starts from 0.
2. contains() ⇒ searches the sequence of characters in string.
   It returns true if sequence of char values are found in string otherwise returns false.
3. getChars() → copies content of this String into specified char Array.
   4 arguments are passed.

```
e.g. public static void main(String[] args) {
       String str = new String("......");
       char[] ch = new char[10];
       str.getChars(11, 17, ch, 0);
       sysout(ch);
     }
```

(endIndex, startIndex of Array, startIndex, Array)

// throws StringIndexOutOfBoundsException if startIndex > endIndex

4. indexOf() → returns index of given character value or substring.
   If it is not found, it returns -1.
5. replace() → returns a string replacing all old char or charSequence to new char or charSequence.
6. toCharArray() → converts string into character Array.

```
e.g. String st: "Hello"
     char[] ch = st.toCharArray();
     for(int i=0; i<ch.length; i++)
       { sysout(ch[i]); }
```

7. toLowerCase() → converts to lowercase
8. toUpperCase() → converts to UPPERCASE
9. trim() → returns a copy of invoking String from which any leading and trailing whitespace has been removed.
10. endsWith() { return Boolean
11. startsWith()

// Java StringBuffer class:

Used to create mutable (modifable) string.

Important constructors :-

1. StringBuffer() : creates empty string buffer with initial capacity of 16.
2. StringBuffer (String str): creates StringBuffer with specified string.
3. StringBuffer (int capacity): creates empty StringBuffer with specified capacity as length.

Important methods:

1. append (String s): appends specified string with this string. It is overloaded like append(char), append(boolean), etc.
2. insert (int offset, String s): inserts specified string with this String at specified position. It is overloaded with like insert(int, char), insert(int, boolean), insert(int, int), etc.
3. replace (int startIndex, int endIndex, String str): replace string from specified startIndex and endIndex.
4. delete (int startIndex, int endIndex): delete from specified.
5. reverse (): reverse the string
6. capacity (): return current capacity

// Diff b/w String & StringBuffer:

String	StringBuffer
1. Immutable	1. Mutable
2. Is slow and consumes more memory when you concat too many strings because every time it creates new instance.	2. is fast and consumes less memory when you concat strings.
3. Overrides equals() method of Object class. You can compare contents of two strings by equals() method.	3. Doesn't override equals() method.

StringBuffer	StringBuilder
1. It is synchronized. ie. thread safe. It means two threads can't call methods of StringBuffer simultaneously.	1. StringBuilder is non-synchronized ie. not thread safe.
2. StringBuffer is less efficient than StringBuilder.	2. More efficient.

METHODS IN JAVA

- A java method is a collection of statements that are grouped together to perform an operation.
- Methods are timesavers and help us to reuse the code without retyping the code.

Method definition:-

```
modifier returnType nameOfMethod (parameter list)
  { // method body }
```

Main components:-

1. Modifier: Defines access type of method.
   - i. public :- accessible in all class in your application.
   - ii. protected : accessible within the class in which it is defined and in its subclasses.
   - iii. private : accessible only within class it is defined in.
   - iv. default ⇒ accessible within same class and package within which its class is defined.
2. Return type : datatype of value returned by method or void.
3. Method Name : camel convention (first letter of each word capital except first word. e.g. enterFirstName)
4. Parameter List: comma seperated list of input parameters are defined, preceded with their data type, within enclosed parantheses.
5. Method Body: code to be executed

e.g.

```
public int max (int x, int y)
  { if (x>y)
      return x;
    else
      return y;
  }
```

- Java methods are of four types :-
  - Methods without return type and without parameters
  - Methods without return type and with parameters.
  - Methods with return type and without parameters
  - Methods with return type and with parameters

- In java methods, parameters accept arguments with three dots.
- These are known as variable arguments.
- once variable arguments are used as parameter method, while calling we can pass as many number of arguments to this method (variable number of arguments), or you can simply call this method without passing any arguments.

```
void demoMethod (String... args)
{ for(String : args)
  { sysout(arg); } }
public static void main (String args )
 { new Sample().demoMethod ("ram", "rahim", "robert");
   new Sample().demoMethod("abc", "xyz");
   new Sample().demoMethod();
 }
```

- Java does not support concept of default parameter, however, you can achieve this using Method Overloading :-
  Using method overloading if you define method with no arguments along with parameterized methods, then you can call a method with zero arguments.

Final methods : If a class declared as final then by default all of methods present in that class are automatically final but variables are not.

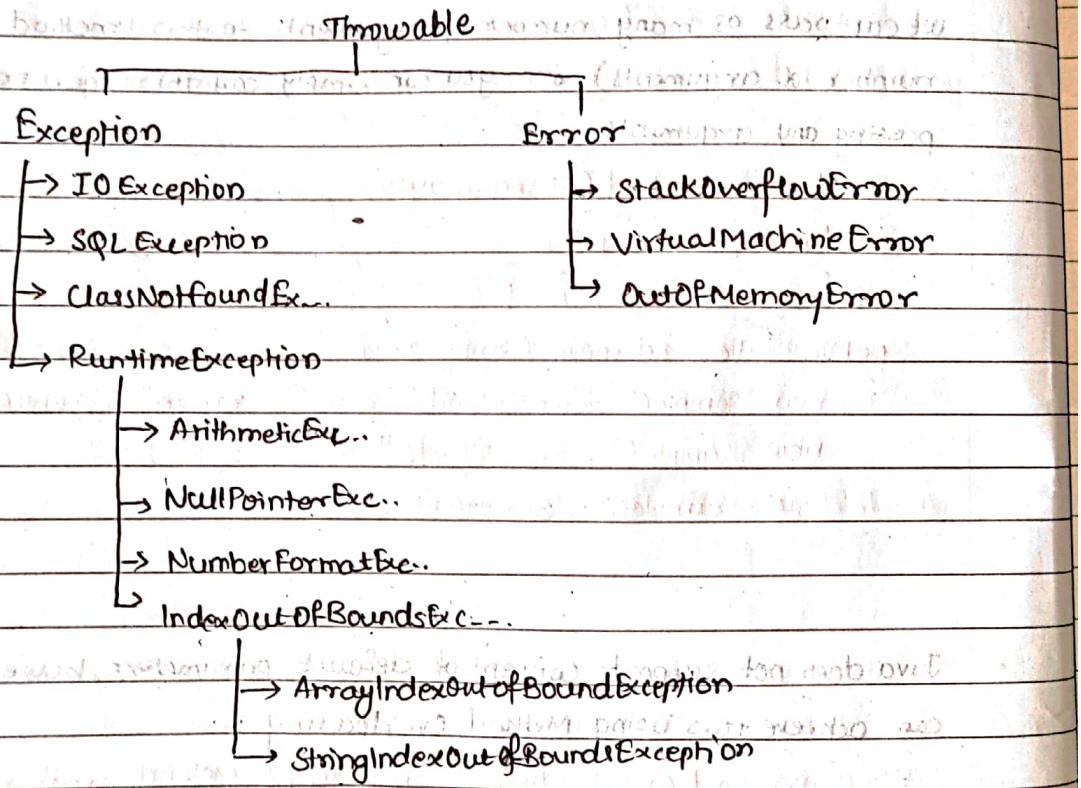
// Recursion in Java :- Method calls itself continuously (repeatedly).
It makes code compact but complex to understand.
Recursion should stop using terminating condition otherwise infinite call of method and program will not stop.

Exceptions in JAVA.

An exception is an unwanted or unexpected event, which occurs during the execution of program, that disrupts the normal flow of program's instructions.

error - indicates a serious problem that a reasonable application should not try to catch.

exception - conditions that a reasonable application might try to catch



- Types of Java Exceptions - Checked, Unchecked, Error (irrecoverable) (checked exception)

Classes which inherit Throwable class except RuntimeException ← Checked at compile Time

↳ inherit RuntimeException Checked at runtime.

e.g.
int a = 50/0 → Arithmetic
String s = null; sysout(s.length()); → Null Pointer
String s = "abc"; int i = Integer.parseInt(s); → NumberFormat-
int[] a = new int[5]; a[10] = 50; → ArrayIndexOutOfBounds

- Exception Handling.

try - block to put exception code. Followed by catch or finally.
catch - handle the exception
finally - execute important code of program. It is executed even if an exception is handled or not handled.
throw - to throw an exception
throws - used to declare exceptions. specifies that there may occur an exception in the method. Used with method signature.

In a method there can be more than one statements that may throw exceptions. put all in try block and provide seperate exception handler within own catch block.
If an exception occurs in try block, exception is handled by exception handler associated with it. Each catch box is an exception handler that handles the exception of the type indicated by its argument. The argument, ExceptionType declares type of Exception that it can handle and must be name of the class that inherits from Throwable class.

There can be only one finally block. Finally block is optional. If exception occurs it will be executed after try and catch blocks. If exception does not occur it will be executed after try.

- User defined Exceptions

Create own exception class and throws that exception using throw keyword. Can be done by extending the class Exception.

```
class JavaException {
public static
e.g  void main () {
    try { throw new MyException(2); }
    catch (MyException e).
    { sysout(e); } }}
class MyException extends Exception {
    int a;
    MyException (int b)
    a=b;
    public String toString() { return ("Exp no." + a);
```

output :-
Exp no. 2.

// Wrapper Classes in Java

- Convert primitive data types to Java objects.
- Java supports only call by value. So, if we pass primitive value, it will not change the original value.
  But if we convert primitive value in an object, it will change original value.
- java.util package provides utility classes.

primitive tye - wrapper class

boolean - Boolean char-Character int-Integer byte-Byte
float-float double-Double short-short long-Long

- AutoBoxing - automatic conversion of primitive data type to into its corresponding wrapper class.

e.g. int a=20;
Integer i = Integer.valueOf(a); //explicit convert
Integer j = a; //autoboxing

- Unboxing - automatic conversion of wrapper class to data type
  Reverse of autobox

e.g. Integer a = new Integer(3);
int i = a.IntValue(); // explicit
int j = a; // unboxing

- Custom wrapper class

```
class customWrapper
{ private int i;
  customWrapper() // default constructor
  {}
  customWrapper(int i) // constructor
  {this.i =i;}
  public String toString() {
    return Integer.toString(i);
  }
}
```

```
//testing wrapper class
public class Main {
public static void main(String[] args)
 {
customWrapper j= new customWrapper();
System.out(j) //autoboxing
 }
}
```

// Java Arrays.

- It is a utility class to perform various common operations on arrays in Java.
- part of Java collections framework
- Sorting, searching, comparing arrays.

// import java.util.Arrays.List

- asList method → returns a list made by specified array.

e.g.

```
String [] strings = {"one", "two", "three"};
List<String> list = Arrays.asList(strings);
sysout(list);
```

OUTPUT
[one, two, three]

- Java arrays SORT // import java.util.Arrays //

sort method → sorts elements, sorts given range of elements.

Sort char

```
char [] chars = {'B', 'D', 'C', 'E', 'A'};
Arrays.sort(chars);
for(char character : chars) sysout(character + " "); // A B C D E
```

Sort INT

```
int [] integers = {
Arrays.sort(integers);
```

For specified range

```
int [] ints = {5, 2, 1, 4, 3, 9, 6, 8, 7, 10};
int fromIndex = 2; int toIndex = 7;
Arrays.sort(ints, fromIndex, toIndex);
for(int i : ints) sysout(i + " "); // 5 2 1 3 4 6 9 8 7 10.
```

- Java Arrays BINARY SEARCH → uses binary search algorithm to search specified value from elements of array / range of array.

```
int [] integers = {5, 2, 1, 4, 3, 9, 6, 8, 7, 10};
int index = Arrays.binarySearch(integers, 2);
if(index >= 0) sysout(index); // 1
else sysout("not found");

int from = 2; toIndex = 7
int index2 = Arrays.binarySearch(integers, fromIndex, toIndex, 9);
if(index2 >= 0 sysout(index) // 5
else sysout("not found");
```

- Java arrays EQUALS.
  Compare two arrays of same type and returns Boolean result.

```
int [] a1 = {1,2,3};
int [] a2 = {1,2,3};
boolean equal = Arrays.equals(a1,a2);
sysout(equal) // true.
```

- Comparing Nested Arrays.

```
int [] a1 = {1,2,3};
int [][] a2 = {1,2,3};
Object [] b1 = {a1};
Object [] b2 = {a2};
boolean equal = Arrays.equals(b1,b2);
sysout(equal)  // false because equals methods is not able to compare nested arrays so have to use deepEquals method.
```

- Java Arrays deepEquals method.

```
int [] a1 = {1,2,3};
int [] a2 = {1,2,3};
Object [] b1 = {a1};
Object [] b2 = {a2};
boolean equal = Arrays.deepEquals(b1,b2);
sysout(equal) // true.
```

OOP - Java.

Java uses object oriented programming paradigm.

It is made up of object and classes.

An object in Java is the physical as well as a logical entity.

whereas a class in Java is a logical entity only.

An entity that has state and behaviour is known as object.

It can be physical or logical (tangible and intangible)

e.g. of intangible object is banking system.

- Characteristics of an Object

State: represents the data (value) of an object. (Attributes)

Behaviour: represents the behavior (functionality) of an object. (Methods)

Identity: An object identity is typically implemented via a unique ID. The value of ID is not visible to the external user. However, it is used internally by JVM to identify each object uniquely.

// Object is an instance of class.

Object definitions:-

object is a real-world entity.
a runtime entity.
an entity which has state and behavior.
an instance of class.

// Class is a group of objects which have common properties.

It is a logical entity, can't be physical.

A class in Java contain:

Fields (Data)
Methods
Constructors
Blocks
Nested Class and Interface

Syntax:

```
class <class-name>
{
field;
method;
}
```

// Core Concepts

- Instance variable

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as instance variable.

- Method

Method is like a function which is used to expose the behavior of an object.

Advantage of method:- • Code Reusability
• Code Optimization

• new keyword

The new keyword is used to allocate memory at runtime.
All objects get memory in Heap memory area.

Class - Object Examples.

1) main method outside class (inside another class)
2) main method inside class

(1)
```
class student {
    int id;
    String name;
}

class test {
    public static void main() {
        student s1 = new student();
        sysout (s1.id);
        sysout (s1.name);
    }
}
```
← defining fields
← creating object
← accessing member through reference variable

(2)
```
class Student {
    int id;
    String name;
    public static void main() {
        Student s1 = new Student();
        Student s2 = new Student();
        Sysout(s1.id);
        Sysout(s2.id);
    }
}
```

3) static

inside class
```
class Student {
    static int id;
    static String name;
    public static void main(String[] args) {
        Student s1 = new Student();
        s2 =          ();

        sysout (id);
        sysout (name);
    }
}
```

Outside class
```
sysout (Student.id)
sysout (Student.name)
```

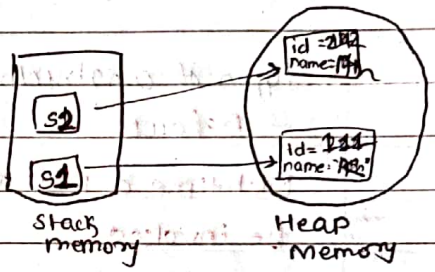
Initialization through reference.

```
s1.id = 04;
s1.name = "Ath";
sysout (s1.id + " " + s1.name)
```

// Initialization through method.

```
class Student {
  private int id;
  private String name;
  public void setValue(int r, String n){
    id = r;
    name = n;
  }
  public void displayInfo(){
    sysout(id + name);
  }
class Test {
  public static void main(){
    Student s1 = new Student();
            s2 =
    s1.setValue(111,"Karthik");
    s2.setValue(222,"Ath");
    s1.displayInfo();
    s2.displayInfo();
  }
}
```

Memory Allocation to object



// Anonymous object

Anonymous means nameless.

An object which has no reference is known as an anonymous object.

It can be used at the time of object creation only.

If object is going to be used only once, anonymous object is good approach.

new Calculation(); ← anonymous object

Calling method through reference

```
Calculation c = new Calculation();
c.fact(5);
```

through Anonymous object

```
new Calculation().fact(5);
```

// Multiple objects by one type

Initialization of reference variables:

```
Rectangle r1 = new Rectangle(), r2 = new Rectangle();
```

Constructors in JAVA.

- special method used to initialize an object
- every class has a constructor implicitly or explicitly.
- If we don't declare a constructor in class, JVM builds a default constructor for that class.
- Same name as the class name
- No explicit return type
- Cannot be abstract, static, final
- can be parameterized or non-parameterized.

Syntax: className (parameters) {

statements;

}

Types of constructor

- Default
- Parameterized

Each time a new object is created at least one constructor will be invoked.

Constructor Chaining

To call constructor from another constructor this keyword is used. this keyword is used to refer current object.

Private constructors

- Prevent class from being instantiated.
- By declaring pvt constructor, it restricts to create object of that class
- Only one object can be created and the object is created within class and also methods are static.
- A class which has a private constructor and methods are static then it is called Utility class.

Copy constructors : Special type of constructors that enables to get copy of existing object.

~~Constructors with different parameters list~~

Take only one parameter which is reference to same class.

Constructor overloading

Creating constructors ~~with~~ different no. of arguments.

* Encapsulation: - enable programmer to group data & subroutines that operate on them together in a place and hide them irrelevant details from users of an abstraction. Bundling of methods & data. Prevent unauthorized parties to directly access them

* Inheritance in JAVA.
- Special feature of OOPP
- one object inherits state and behaviour of parent object
- Using inheritance, one can create new classes that are built on existing classes
- When class is inherited from existing class, class can reuse methods and fields of parent class
- Child class can add new methods and field in it
- Inheritance represents IS-A relationship which is also called parent-child relationship
- Inheritance is used for - Method Overriding
  Reusability of code

- Terms:-

Class - group of objects which have common properties

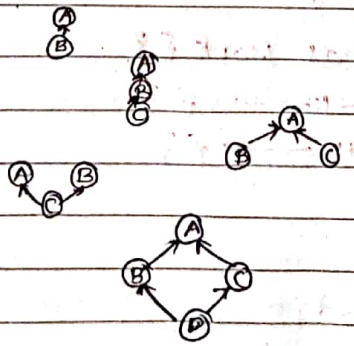
subclass/child class :- Inherits other class. Derived/Extended class.

Super/parent class :- Class from where subclass inherits the features. Base class.

Reusability:- It is a mechanism which facilitates you to reuse the fields and methods of existing class when you create a new class. You can use same fields and methods already defined in previous class.

Syntax:- class SuperclassName { ... }
class subclass extends superclassName { }

Types of Inheritance
1. Single
2. Multilevel
3. Hierarchical
4. Multiple
5. Hybrid.



Multiple inheritance is not supported through class in Java, it is possible by Interface.

* Method Overloading.
Creating methods with same name but different no. of arguments or data types.

Method Overriding

Redifining a method in a subclass which has been defined in its parent class. Must have same name as that of in parent class, same parameters

Super Keyword.

super.name

- Used to refer immediate parent class instance variable.
- Used to invoke parent class method → super.name();
- Used to invoke parent class constructor. → super();
- Super() is provided by the compiler implicitly.

* POLYMORPHISM IN JAVA

- Single action in different way
- Poly-morphs = many-forms
- Compile Time, Runtime.
- Overloading of static method is compile time
- Runtime also called as dynamic method dispatch.
- Overriding of methods are used in runtime polymorphism.
- Runtime polymorphism uses upcasting in inheritance

Upcasting

```
class A{ }
class B extends A{ }
A a = new B();
```

Reference variable of parent class → object of child class

* run time polymorphism e.g.

```
class Bank{ }
class SBI extends Bank{ }
class ICICI ——"—— { }
class HDFC ——"—— { }
class Main{
    Bank b;
    b= new SBI();
    b= new ICICI();
    b= new HDFC();
```

Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime. Determination of method to be called is based on object being referred to by reference variable.

* Runtime polymorphism with Data Members
- A method is overriden, not the data members
- Runtime polymorphism cannot be achieved by data members

```
class Bike { int speed = 90; }
class Honda extends Bike { int speed = 150; }
void main {
    Bike obj = new Honda();
    sout(obj.speed); ⟶ 90
}
```

* Static Binding & Dynamic Binding
- Connecting a method call to the method body is known as binding.
- There are two types of binding
  - Static Binding (Early Binding)
    type of object determined at compile time.
  - Dynamic Binding (Late Binding)
    type of object determined at run time

Type
int data = 30; type of variable
Dog d1; type of Dog (Type of reference)
Dog d1 = new Dog(); type of object.

Static.

```
Dog d1 = new Dog();
d1.eat();
```

Dynamic.

```
class Animal
class Dog extends Animal
void main{
    Animal a = new Dog();
    a.eat(); }
```

Object type cannot be determined by compiler because instance of Dog is also an instance of Animal So compiler doesn't know its type, only its base type.

instanceof Operator

```
class Animal{}
class Dog extends Animal{}
void main{
    Dog d = new Dog();
    Animal a = new Animal();
    sout(d instanceof Animal); true
    sout(a instanceof Dog); false
```

Downcasting → act of casting reference of base class to one of its derived class.

```
class Animal
class Dog extends Animal
void main{
    Dog d = new Dog();
    Animal a = new Animal();
    Animal b = new Dog(); // auto-upcasting.
    Dog c = new Animal(); // compile time error
    Dog c = (Dog) new Animal(); // DOWNCASTING.
```

e.g.

```
class Animal{}
class Dog extends Animal{
    static void method(Animal a){
        Dog d = (Dog)a; // Downcast
        sout("Perform doowncast");
    }
    static void main(){
        Animal a = new Dog(); // Upcast
        Dog.method(a);
    }
}
```

* Abstract class

- class declared with keyword abstract is called abstract class.
- has abstract & non-abstract methods.
- Abstraction is process of hiding implementation details and showing only functionality to the user
- ways to achieve abstraction:-
  1. Abstract class
  2. Interface.
- Abstract class cannot be instantiated.
- It can have constructors & static methods also
- It can have final methods which will force subclass not to change the body of the method.

abstract class
abstract class A {}

abstract method
abstract void print(); has no body.

Example of abstract class that has constructor, abstract & non-abstract methods.

```
abstract class Bike {
    Bike() { sout bike created };
    abstract void run();
    void changeGear() { sout gear changed };
}
class Honda extends Bike {
    void run() { sout running };
}
class Main {
    public static void main (String [] args) {
        Bike obj = new Honda;
        obj.run();
        obj.changeGear();
    }
}
```

OUTPUT: -
bike created
running
gear changed.

- If there is an abstract method in a class that class must be abstract
- If you are extending an abstract class that has an abstract method you must either provide implementation of method or make this class abstract.

* INTERFACE :-
- Type of unimplemented class.
- Has static constants and abstract methods.
- Mechanism to achieve abstraction
- There can only be abstract methods in Java interface, not method body.
- Used to achieve multiple inheritance
- represents IS-A relationship.
- cannot be instantiated just like abstract class.
- one can have default, static and private methods in an interface
- can be used to achieve loose coupling

syntax:-

```
interface <interface_name>
{
    // declare constant fields
    // declare methods that abstract
}
```

- Java compiler adds public and abstract keywords before interface method.
- Adds public, static and final keywords before data members

class	interface	interface
↑ extends	↑ implements	↑ extends
class	class	interface

multiple inheritance:-

```
interface Printable { void print(); }
interface Showable { void show(); }
class A implements Printable, Showable {
    public void print() { sout print }
    public void show() { sout show }
    void main {
        A obj = new A();
        obj.print();
        obj.show();
    }
}
```

Print
Show

Interface Inheritance

A class implements an interface, but one interface extends another interface.

```
interface print { void print(); }
interface show extends print { void show(); }
class A implements show { void print();
                          void show(); }
```

*Default Method in Interface

```
interface Drawable {
    void draw();
    default void msg(){ sout default }
}
class Rect implements Drawable {
    public void draw(){ Drawing }
}
class Main{
    void main(){
        Drawable d = new Rect();
        d.draw();
        d.msg();
    }
}
```

drawing
Default

- Static Method in Interface

```
interface Drawable {
    void draw();
    static int cube(int x){ return x*x*x; }
}
class Rect implements Drawable {
    public void draw(){ sout drawing }
}
class Main{
    void main(){
        Draw d = new Rect();
        d.draw();
        t = Drawable.cube(3);
        sout(t);
    }
```

Drawing
27.

Difference between abstract class and interface:

Abstract class	interface.
• has abstract & non-abstract methods	• has only abstract. can also have default & static
• doesn't support multiple inheritance	• supports multiple inheritance
• can have final, non-final, static, non-static variables.	• only static & final
• can provide implementation of interface	• can't provide implementation of abstract class
• abstract keyword to declare	• Interface keyword
• can extend another Java class & implement Java interface.	• can extend another interface only
• can be extended using keyword extends	• can be implemented using keyword implements
• can have members like private, protected	• members are public by default