**JAVA Contents**

1. Project, package, Class, object, method
2. Variables, data types, return type and wrapper class.
3. Access Modifiers
4. Control statements/Looping statements
5. String, String Buffer, String Builder
6. Constructor, this, super
7. Static
8. Exception Handling
9. Array
10. Collections
11. OOPS
    1. Inheritance
    2. Polymorphism
    3. Abstraction and Interface
    4. Encapsulation
12. File

Throw, Throws and throwable, user defined exception, final

Typecast, random num gen, regex, class scope restriction

File write config property

JDK/Eclipse – 64bit download

Install jdk and open eclipse.

**Maven**:

Maven is a project management tool.

Downloading the required jar files using the dependencies.

It is the central repository for all the dependencies.

Dependency management

Folder structure.

  Project object model popularly known as POM.

**JAVA**

Project ->

Packages: - **c**amel Notation

Collection of similar classes

**Class**: - **P**ascal notation

A Class is a collection of object and methods which has similar properties.

Eg: In theatre based on the ticket price, we can categorize them by class as Fistclass, Elite, balcony.. etc.

**Method**:

A **method** is a set of code grouped together to perform a certain task or operation.

Variable:

If we want to access the instance variable from the main method we need to create an object for the class.

Eg: obj.variable name

Data types:

Primitive:-

<https://static.javatpoint.com/images/java-data-types.png>

// Integer

// allow till 3 digits only

**byte** age = 32;

System.***out***.println(age);

// allow till 5 digits only

**short** dob = 31089;

System.***out***.println(dob);

// allow till 9 digits only

**int** x = 997647475;

**int** birth = 3101989;

System.***out***.println(birth);

// Long

**long** ph = 9976474750l;

System.***out***.println(ph);

// Decimal

**float** salary = 40000.00f;

System.***out***.println(salary);

**double** sal = 40000;

System.***out***.println(sal);

// Char

**char** gender = 'm';

System.***out***.println(gender);

// String - Non Primitive data type

String name = "Kiruthiuvasan";

System.***out***.println(name);

//Boolean

**boolean** passport=**true**;

System.***out***.println(passport);

Default values:

Integer type – 0, decimal – 0.0, char and String – null and Boolean = false

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

**Control/Control flow Statements**

A statement which has control over the loop or program is control statements.

The Conditional Statements/decision-making statements (if, if else, switch), the looping statements (for, while, do-while), and the branching statements (break, continue, return) supported by the Java programming language.

Conditional statements:

If

If.. else

If.. elseif.. else

If{ if{} .. else{}} else{} –nested if

Switch case:

After the condition satisfies, If we don’t include break, it will start printing all the following cases without checking condition

Data type var=;

Switch(){

case:

s.o.p;

break;

case:

s.o.p;

break;

default:

s.o.p;

}

Break: It will break the loop, when the condition satisfied

Continue: It will skip the particular iteration and keep execute with next step, when the condition met.

System.exit(0): It will terminate entire program, when the condition met.

**String**

String is sequence of characters. A Non-primitive datatype. Also a class.

Mutable – Changeable

New memory space will be created for each object. It won’t overwrite and will change the pointer to the new object.

Immutable – Changeable

No new memory will be created for each object. It will overwrite the existing.

String is immutable and constant, can’t be changed once it created.

String class is present in java.lang package.

String creation:

1. String literal

String name = “Kiruthivasan”;

If String 1 and String 2 have the same value, it will not create a separate space for 2nd string value. It will point the string2 to string 1’s memory address

String a=”India”;

String b=”India”;

String 1

“Kiruthivasan”

String 2

String literal is good at memory efficient

1. Using new

String name=new String(“Kiruthivasan”);

Here we have created an object name for class String

Here we r mentioning as new, so it will create a new memory space for String2’s value, even though the values of string 1 and 2 are same.

String 1

“Kiruthivasan”

String 2

“Kiruthivasan”

Length = last index position+1

Usually beginning index included and end index won’t included

String, String buffer and String builder are implementing the CharSequence interface

**String Buffer**

String Buffer is mutable

It is synchronized and won’t support multi-threading. Actions happen sequentially, so it is called thread safe.

No String literal option, can be created only using new keyword

String – can’t be reversed. Need to write logic

StringBuffer – can be reversed using reverse()

StringBuilder – can be reversed using reverse()

**String Builder**

Mutable

Asynchronized, it’s not following sequential order. It supports multi-threading

No String literal option, can be created only using new keyword

|  |  |  |
| --- | --- | --- |
| String | String Buffer | String Builder |
| Immutable | Mutable | Mutable |
| No reverse method | Reverse() | Reverse() |
|  | Synchronized | Asynchronized |
|  | Thread safe | No thread safe |
|  | Less efficient due to the sequential process | More efficient due to multiple access at the same time |

**Constructor**

A constructor in Java is **a special method that is used to initialize objects**.

Constructor is a block of code. { }

Properties:

Class name and constructor name should be same.

No explicit return type

It can’t be abstract, static, final and synchronized

Types:

No argument

Parameterized

**This/Super Keyword**

This keyword is used a reference variable. Using to refer the current object

**Static**

Static is a keyword. Can be used with class, method, variable and block.

No need to create an object.

We can call a static method from non-static method, but we can’t call non-static method from a static method.

**Exception Handling**

An unexpected event occurred during the execution of a program, and interrupting the flow.

Error is different from exception. Exception can be handled by programmer, but error is a severe one it can’t be handled.

To handle the exception, we can write the logic programmatically that is exception handling

Throwable

Exception Error

Exception types:

1. Checked exception/compile time exception
2. Unchecked exception/run time exception

Try:

Writing the code inside the block

Catch:

If any exception occurred, will handle in this catch block

Syn:

try{

code/logic

}catch(Exception e){

e.printStackTrace(); - will print the exception

exception handling code

}

There can be multiple catch block

If theres no exception, catch block will not get executed

Finally:

This is a block of code which will get executed, even the exception occurs or not.

Mostly it will contain – db close

Nested try/catch – similar to nested if..else

try{

code

try{

code

}catch(){

}

}catch(){

}

**Array**

Array is used to store the multiple values of similar data type. It is an index based one, starting from 0 to n-1.

It allows duplicate.

Static memory allocation.

Disadv:

1. Fixed size
2. Storing only similar data type

Length:

variable.length;

Arrays.Sort(variable);

ArrayUtils.reverse(var);

//First index

**int** f = marks[0];

//last index

**int** l = marks[marks.length-1];

System.***out***.println(l);

//Middle index

**int** m = marks[marks.length/2];

System.***out***.println(m);

Array copy

<jdk 1.5

String name[]= {"Kiruthivasan","Lakshmi","Shravan","Sridhar","Shravan","Shyam"};

String dupe[]=**new** String[4];

System.*arraycopy*(name,0,dupe,0,4);

>jdk 1.5

**int** a[]= {91,79,88,81,71};

**int** b[]=Arrays.*copyOfRange*(a, 0, 3);

**Collection**

A collection is group of individual objects. Its used to store the multiple values of dissimilar datatypes in a single reference variable.

Memory waste is low and run time memory allocation.

Types:

List

Set

**List**

It is an interface. Index based one. Allows duplicated

**ArrayList**:

It’s a class implementing List interface

Its implementing Random Access interface, so that suitable for search and retrieval operation. Not suitable for insertion deletion, due to element position displace it will take lot of time.

Maintains insertion order

List<String> item=new ArrayList<String>();

**LinkedList**

It’s a class implementing List interface

Its NOT implementing Random Access interface, so not suitable for search. Internally used doubly linked list, element pointing to next element, so no value displacements.

Suitable for Insertion/Deletion.

Maintains insertion order

List<Integer> marks=**new** LinkedList<Integer>();

SET

SET is an Interface, it’s value based one. Won’t allow duplicate.

It’s a class implementing SET interface.

**Hash** **Set**

ImplementsSet interface

Follows random order

**Linked Hash Set**

ImplementsSet interface

Follows Insertion order

**TreeSet**

ImplementsSet interface

Follows Ascending order

**Map**

Map is not a true collection, since it’s not implementing anything from Collection

It represent data in Key, value pair.

Values can be fetched through Key.

Key - Unique

Value – Allow duplicate

If Key is duplicated in code, it won’t show compile/run time exception, it will just update the latest entry to that key. Same for null key. Only Tree Map won’t allow null key.

Put method is used to add elements in the map

***Hash Map***

***Linked Has Map***

***Tree Map***

Hash Map:

Allows unique null key

Follows Random order

Copy in to another Map – putAll

Clear values – Clear

To Check if the Key is present – ContainsKey

To check if value is present – Containsvalue

Synatx to print key, value:

**for**(Map.Entry<String, Integer> n: m.entrySet()) {

System.***out***.println(n);

}

Linked HashMap:

Follows Insertion order

Tree Map:

Follows Ascending order

It won’t allow null Key entry, due to need to compare keys to sort.

**OOPS**

Inheritance

Polymorphism

Abstraction

Encapsulation

**Inheritance**

Inheritance is accessing one class property from another class. Using the keyword extends we are achieving inheritance.

Types:

1. Single
2. Multilevel
3. Multilple
4. Hierarchical

**Single**

Relationship of 1 parent and 1 child class.

Parent class extends Child class

**Multilevel**

A child class accessing the property of parent class, which is a child acquired properties from its parent.

It’s called inheritance chaining. Child class accessing the property in a tree level structure.

**A**

**B**

**C**

**Hierarchical**

More than 1 child classes accessing the parent class.

**A**

**B C**

**Multiple Inheritance**

Child class accessing the property from more than 1 parent at a time.

In java multiple inheritance can’t be achieved using extends keyword.

If two parents class have same methodname with action, the code will get confuse which class logic to take first. It will create ambiguity problem. Compile time err will throw.

Multiple inheritance can be achieved through *Interface*.

**Polymorphism**

Executing methods in more than 1 form.

Method Overloading –also called as compile time polymorphism/static binding/early binding

Method Overriding – also called as runtime polymorphism/dynamic binding/late binding

**Method Overloading:**

Same method name, same class and argument should be different.

Arguments:- different data types, different number of arguments, different order of arguments.

**Method Overriding** :

Same method name, same argument, but different class.

**Abstraction**

Abstraction means hiding the implementation/logic and showing only the functionality.

0 to 100% Abstraction is Abstraction, also called partial abstraction.

Contain both abstract and non-abstract methods.

If a class having the unimplemented method, then the method should be declared as abstract method.

If a class is having at least 1 method, then it should be declared as abstract class.

Child class can access abstract class using keyword extends.

If child class doesn’t implement all the abstract method, syntax err will throw to implement that missed abstract method/s or to change the child class to Abstract class.

We can’t create an object for an abstract class.

Abstract class may contain only concrete methods without abstract methods. So if a class is created with abstract then it is just called as abstract class.

Allow public, protected and default not private

public abstract class ClassName{

public abstract void methodName();

}

**Interface**

Interface is 100% abstraction, contain only abstract methods.

Should be created as Interface instead of class.

Interface can be implemented by a class using the keyword implements.

If child class doesn’t implement all the abstract method, syntax err will throw to implement that missed abstract method/s or to change the child class to Abstract class.

We can’t create an object for an abstract class.

A class can inherit multiple interfaces, in this way *multiple inheritance* is achieved.

Eg: **public** **class** ViceCity **implements** GTA, Game{

}

Allow public only, if don’t give any access modifier, it will take as public not as *default* access modifier.

Interface extends Interface

Class extends Class

Class implements Interface

**Encapsulation**

A process of groping the data members and corresponding methods in to a single unit called Encapsulation.

Eg: Class Student

Public class Student{

//Data members

String stud\_name;

int marks;

int rollNo;

//Methods

readMarks()

WriteMarks()

}

**FILE**

File Writer:

**package** fileOperations;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** FileWriteLearn {

**public** **static** **void** main(String[] args) **throws** IOException {

String loc="MyFirstFile.txt";

String con="I am Kiruthivasan";

FileWriter fw=**new** FileWriter(loc);

fw.write(con);

fw.close();

}

}

//To increase the performance we r using Buffered writer

**package** fileOperations;

**import** java.io.BufferedWriter;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** BufferedFilleWrit {

**public** **static** **void** main(String[] args) **throws** IOException {

String loc="MysecFile.txt";

String con="I am Kiruthivasan working as Tester";

FileWriter fw=**new** FileWriter(loc);

BufferedWriter bw=**new** BufferedWriter(fw);

bw.write(con);

bw.close();

}

}

FileOutputStream – Usage is write the content as its without any changes/encoding

**package** fileOperations;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** FOS {

**public** **static** **void** main(String[] args) **throws** IOException {

String loc="UsingFos.txt";

String con="I am Kiruthivasan the great";

FileOutputStream fw=**new** FileOutputStream(loc);

**byte**[] bytes = con.getBytes();

fw.write(bytes);

fw.close();

}

}

**File Reader**

Single line:

package fileOperations;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class FileBufferReader {

public static void main(String[] args) throws IOException {

String loc="MyFirstFile.txt";

FileReader read= new FileReader(loc);

BufferedReader br=new BufferedReader(read);

String readLine;

System.out.println(readLine);

br.close();

}

}

Multiple Line:

package fileOperations;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class FileBufferReader {

public static void main(String[] args) throws IOException {

String loc="MyFirstFile.txt";

FileReader read= new FileReader(loc);

BufferedReader br=new BufferedReader(read);

String readLine;

//Reading multiple lines

while((readLine= br.readLine())!=null) {

System.out.println(readLine);

}

br.close();

}

}