**NOTES:**

1. **Static functions** can only access static stuff. **Non-static**/ **Virtual** functions can access **static** and **non-static/virtual** stuff.
2. **Static variables** cannot be declared inside the method. It has to be declared **inside the class** & **outside the method.**
3. If you want to access a **static (variables/methods)** defined in another class in the same or different file, you should use **className.variableName** OR **className.MethodName()**
4. To access the **non-static/virtual** stuff (variables/methods) in other class files, you must create an object of that class & use the object reference of that class to access the stuff.
5. **Local variables:** These are the variables those are **declared inside a method**. Life of local variable is inside a method only. Local variables must be initialized (int var = 0) else it throws an error when it is used.
6. **Global variables:** These are present inside the class but outside the method. Global variables **need** **not** to be initialized, non-initialized variables has value of 0. If you change the value of a global variable**(static/non-static or virtual)** using any one of the object of that class, the updated value will be available to all the objects of that class.
7. **Static** variables are common for all the objects of a respective class.
8. One class file can have multiple classes but only one Public class per java file.
9. Creating an obj of class?

**Car c1 = new Car();**

- Here, **c1** is a reference to the object & **"new Car()"** is an actual object & memory location will be allocated for that object and the object reference points to the memory location. All the non-static stuff (i.e. Non-static variables and non-static methods) resides in the class object.

1. You can initialize static variables in Class directly if you want or you can initialize it inside the method.
2. **Non-static variables** cannot be used without a reference of an object.
3. **Static** is common for all objects, If you change the value of a global static variable using any one of the object of that class, the updated value will be used for that object and all other objects.
4. In main method if you want to access the **non-static** variables in the class, you have to create an **object** of that same class in main method and using that object you can access **the non-static variables. - Ex. c1.functionName()**
5. When java is executing the code, it calls the main method without creating an object of the class.

**Class variable:** These are static variables inside the class.

**Instance variable:** These are non-static variables inside the class.

**Local variable:** These are inside the method.

**Packages:**

1. To access the (variables/methods) in the classes those are in other package, you should always import the respective class in that package into current class file using statement "import packageName.ClassFileName;"...(.java file)
2. You will only be able to import the public classes from the other packages & can access only public stuff(i.e. public variables/ methods) in the imported classes.

**Object and Object ref:**

1. An object is a **collection** of all **non-static** stuff in the class.
2. Static variables are common for all objects of that class, If you change the value of a global static variable using any one of the object reference of that class, the updated value will be used for that object and all other objects. However, this is not the case with non-static variables.
3. When java is executing the code, it calls the main method without creating an object of the class.
4. If there are multiple references of an object are pointing to the same memory location, which is possible. When any of the reference changes the value in that memory location, the updated value is available for all other references too.
5. Only static object reference can static Test2 t2 = new Test2(); The objects created inside the class.

**Pass by Value and reference:**

1. Pass by value and Pass by reference. -
2. swap(int a, int b) - pass by value
3. swap(className objRef) - pass by reference - it changes the value of the variable present inside the memory location.

**Constructors:**

1. Constructor has no return type.

2. It help us initialize the object.

3. Constructor is called, as soon as the object of the class is created. It will be called number of times the object of the respective class has been created.

4. Constructor overloading is possible.

5. Constructor / function overloading - Same constructor name or function name but different parameters.

**Example:** Constructor overloading

public car (){

}

public car (int a, int b){

}

**Questions:**

Private class and variables

**this** keyword- "this" is a keyword in java which always point towards the current object.

**this** - is the current instance

**package** testingPackage;

**public** **class** ThisKeyword {

**int** a;

**int** b;

**public** **static** **void** main(String[] arg) {

ThisKeyword th = **new** ThisKeyword();

th.setData(10, 12);

th.showData();

}

**public** **void** setData(**int** a, **int** b) {

**this**.a = a; //sets the value of global variable

**this**.b = b;

}

**public** **void** showData() {

System.***out***.println("Value of A = " + a);

System.***out***.println("Value of B = " + b);

}

}

**Interface:**

1. Interface provides an outline structure
2. Interface describes What the application is suppose to do.
3. In interface, we just declares the functions.
4. You cannot create an object of an interface but you can create objects of classes implementing interfaces.
5. Class always implements an interface
6. In class we need to override the methods in interface & define the methods.
7. In a class that is implementing the interface, you need to define all the methods available in an interface. Or if you want to implement only few methods from an interface, you must create an abstract class instead of regular class.
8. All the variables in an interface are by default **public**, **static** and **final**. It is **mandatory** **to** **initialize** final variables.
9. One class can implement more than one interfaces

public class FortisHospital implements Hospital, Hospital1{

}

**Inheritance:**

* 1. Root of an inheritance actually starts from the interfaces in java. Interface is something that is incomplete. It describes what an application is suppose to do.
  2. Child class(derived/subclass) extends the parent class(Base/ Super) - The extending child class will inherit all the properties of Parent class. In inheritance, extending class/child class is a superset.
  3. One class cannot extends two classes at a time.
  4. When a child class extends Parent, you can access the variables/methods inside the parent class using child class object. However, you cannot access the variables/methods in the child class using parent class's object.
  5. When a child class extends a Parent/Super/Base class, When you create an object of Child class, the control first goes into a default constructor of Parent/Super/Base class and then it comes inside the constructor of child class.
  6. By default, every constructor of any class will have a method super() which will call the default constructor in Parent class.
  7. In JAVA, Every class extends an Object class by default. Hence, every constructor calls the default constructor in Object class.
  8. If you have a class/instance variable with same name in both Parent and Child class, and in specific case, you want to use the value of variable from parent class in the child class, you can use a super keyword. (E.g. **super.varName**)

**Overriding:**

* 1. In the case of inheritance where a child class extends Parent class, When the method with the same name is present in Parent and child classes, it will use the function in the child class and not the function in the Parent class this is known as method overriding.

**Abstract Class:**

1. Using an abstract class, you can implement only few methods from respective interface.

2. You **cannot** create an object of an abstract class. In order to access the stuff in abstract class, you need another class to extend the abstract class.

**Access Modifiers:**

**Public** : Can be accessed from anywhere.

**Private** : You **cannot** access the private Variables/methods outside the class. It can only be accessed within the class. You cannot make the classes private.

**Default** : Only be accessed within the class and classes in the same package.

**Protected:** Only be accessed within the class, classes in the same package and child classes in different packages.

**Exception Handling:**

**public** **class** Division {

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

**try**{

**int** i = 2/0;

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

}**catch**(Exception e){

**System.*out*.println("Exception "+ e.getMessage());**

}

System.***out***.println("Test");

}

}

1. The moment an error comes, the control goes inside the catch block.
2. **Exception & Throwable** are inbuilt classes in JAVA and **e** is an object of that class.
3. **Exception** is useful because even if there is an error, it does not terminate the program. You can report the errors/exception and can continue with the execution.
4. Exceptions usually occurs when the programmers wrote the program incorrectly.
5. You can also use Throwable instead of Exception. It is a super class of Error and exception. you can catch both exception and errors using Throwable.

Ex. }**catch**(Throwable e){

1. No matter if the exception comes or not, the **finally** block will always be executed.
2. Variables declared only in try catch block cannot be used outside the block. If you do so, it would generate compile error.
3. Even if there is an exception in the code, the statements after the Try-catch block will still be executed.
4. If you try to perform any operation on the object that is null, you will get null pointer exception.
5. If a method which is called has throws Exception, then the calling method should also have throws Exception.

**Ex**

**public** **class** Test {

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

**int** a = *divide*(4, 2);

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

}

**public** **static** **int** divide(**int** a, **int** b) **throws** Exception {

**int** result = a / b;

**return** result;

}

}

1. **What are the type of exceptions?**

* **Arithmetic Exception**  
  It is thrown when an exceptional condition has occurred in an arithmetic operation (int i=2/0).
* **ArrayIndexOutOfBoundException**  
  It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
* **StringIndexOutOfBoundsException**  
  It is thrown by String class methods to indicate that an index is either negative than the size of the string
* **NullPointerException**  
  This exception is raised when referring to the members of a null object. Null represents nothing
* **FileNotFoundException**  
  This Exception is raised when a file is not accessible or does not open.
* **NumberFormatException**  
  This exception is raised when a method could not convert a string into a numeric format.
* **NoSuchFieldException**  
  It is thrown when a class does not contain the field (or variable) specified
* **NoSuchMethodException**  
  It is thrown when accessing a method which is not found.
* **ClassNotFoundException**  
  This Exception is raised when we try to access a class whose definition is not found
* **IOException**  
  It is thrown when an input-output operation failed or interrupted
* **InterruptedException**  
  It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.
* **RuntimeException**  
  This represents any exception which occurs during runtime.

1. **What are the type of ERRORS?**

* **OutOfMemoryError**
* **StackOverflowError**
* **LinkageError**

**throw** : Deliberately throws an exception.

**throws :** Checked Exception & Unchecked exceptions

**final :** When a variable is declared with the keyword final, its value cannot be changed. It is **mandatory** **to** **initialize** final variables.

**Checked & Unchecked exceptions:**

1. Checked exceptions are compile time.
2. Unchecked exceptions are runtime.

**Collections:**

1. Clone the Arraylist

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.ListIterator;

**public** **class** Collections {

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

ArrayList<String> list = **new** ArrayList<String>();

list.add("A");

list.add(1, "One");

list.add(2, "Two");

ArrayList list2 = (ArrayList) list.clone();

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

System.***out***.println(list2);

list.clear();

list2.remove(0);

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

System.***out***.println(list2););

}

}

=========================== JAVA INTERVIEW QUESTIONS ==========================

1. Can you extend main method in Java?

2. Explain how hashmap works internally?

3. Difference between StringBuffer & StringBuilder?

**String**

String is immutable ( once created can not be changed )object . The object created as a String is stored in the Constant String Pool. Every immutable object in Java is thread safe ,that implies String is also thread safe . String can not be used by two threads simultaneously. String once assigned can not be changed.

**StringBuffer**

StringBuffer is mutable means one can change the value of the object . The object created through StringBuffer is stored in the heap. StringBuffer has the same methods as the StringBuilder , but each method in StringBuffer is synchronized that is StringBuffer is thread safe . Due to this it does not allow two threads to simultaneously access the same method . Each method can be accessed by one thread at a time . But being thread safe has disadvantages too as the performance of the StringBuffer hits due to thread safe property . Thus StringBuilder is faster than the StringBuffer when calling the same methods of each class. String Buffer can be converted to the string by using toString() method. StringBuffer demo1 = new StringBuffer("Hello") ; // The above object stored in heap and its value can be changed . demo1=new StringBuffer("Bye"); // Above statement is right as it modifies the value which is allowed in the StringBuffer

**StringBuilder**

StringBuilder is same as the StringBuffer , that is it stores the object in heap and it can also be modified . The main difference between the StringBuffer and StringBuilder is that StringBuilder is also not thread safe. StringBuilder is fast as it is not thread safe . StringBuilder demo2= new StringBuilder("Hello"); // The above object too is stored in the heap and its value can be modified demo2=new StringBuilder("Bye"); // Above statement is right as it modifies the value which is allowed in the StringBuilder

4. Print reverse string

6. Count the number of occurrence of a word/ letter in a string./ Code to check the frequency of word occurrence?

7. Difference between Array and ArrayList?

**How to convert Array to ArrayList?**

List<Element> list = Arrays.asList(array);

Here, the

8. Multi dimensional array

9. What is garbage collection in JAVA? --> Unused objects are destroyed by JAVA with the mechanism called garbage collection.

10. How many ways we can define the string in JAVA?

Answer:

**Using String literal**

String str = "Hello!";

**Using new keyword**

This is the common way to create a String object in java.

String str1= new String("Hello!");

**Using character array**

You could also convert character array into String here

char ch[]={ 'H','e','l','l','o','!',};

String str1=new String(ch);

11. Why strings are immutable in java?

12. Difference between **.equals()** and **==**

**Answer:**

1. Main difference between .equals() method and == operator is that one is method and other is operator.
2. We can use == operators for reference comparison (address comparison) and .equals() method for content comparison. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.
3. If a class does not [override the equals method](https://www.geeksforgeeks.org/overriding-equals-method-in-java/), then by default it uses equals(Object o) method of the closest parent class that has overridden this method. See [this](https://www.geeksforgeeks.org/override-equalsobject-hashcode-method/)for detail
4. Coding Example:

|  |
| --- |
| // Java program to understand  // the concept of == operator  public class Test {      public static void main(String[] args)      {          String s1 = new String("HELLO");          String s2 = new String("HELLO");          System.out.println(s1 == s2);          System.out.println(s1.equals(s2));      }  }  13. How to check the data type of variable?  String name = "James";  StringBuilder sb = **new** StringBuilder(name);  // following will return true since name is type of String  **boolean** result = name **instanceof** String;  System.***out***.println(result); |