**Java Notes:**

1. **OOPS concepts:** There are four object-oriented programming principles. We would be discussing each of the principle in details .
   1. **Encapsulation:**

Encapsulation is one of fundamental principle of object-oriented programming. Encapsulation is hiding/protecting data from getting corrupted inadvertent/advertently.

Encapsulation is done by making member variable as private and allowing the data accessible through method/functions this leads to encapsulation of data. Methods/function can help in checking the data for validity hence data is protected.

In java encapsulation achieved by proving getter and setter method and making variables as private.

With encapsulation we can make the data as read only by only providing getter methods.

Here is example of encapsulation in java:

“public class EncapExample{

private int value;

private getValue(){

return value;

}

private int setValue(int v)

if(v>0)

value=v;

}”

* + 1. **What is Encapsulation in Java?**

Ans:

Encapsulation is the main concept of OOPs (Object-Oriented programming). Encapsulation means wrapping or binding data and methods in a single unit. It is also known as data hiding. Data hiding means it derived the result of a specific class. This also means that is not directly accessible. Data in Encapsulation is private, the results of the data are called by using different methods. The code is wrapped in such a way that it cannot be randomly accessed by other codes outside the class.

Encapsulation Rules:

Always make-instance variable private.

Always make public accessor methods and force calling code to these methods instead of directly calling the instance variable.

Use naming convention set() and get for these methods.

package encapsulation;

public class DebitCard {

private int accountno = 789012;

private int pinno = 6543;

private double balanceamount = 200000;

public void withdrawAmount(int accountno, int pinno; int amountwithdraw) {

System.out.printIn("amountwithdraw < balanceamount")

}

}

* + 1. **Q: How Encapsulation is helpful in Java?**

Encapsulated codes can be tested easily. Encapsulation helps in securing the code.

To do changes in the Encapsulated code is easy as we can change in a code or class without affecting other code or classes.

Encapsulated code can be reused and maintained for data hiding.

* + 1. **What are the advantages of Encapsulation?**

Security: It keeps the codes and data safe from external inheritance.

Flexibility: It keeps the codes and data safe from external inheritance.

Easy unit testing: As all data is encapsulated or wrapped in a single unit the code testing becomes easier.

Reuse of code: It allows modifying implemented code without breaking other code that has implemented the code.

* + 1. **Where Encapsulation is used?**

Encapsulation is used in various design patterns and real-life problems that make use of the Encapsulation object-oriented concept. It is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties direct access to them.

* + 1. **Q: What are the advantages of using Encapsulation?**

Ans:

Encapsulation binds or wraps the data and actions in a single unit using variables and methods which makes the use of data simpler and more flexible. Using encapsulation makes it easier to make the changes without affecting different classes. Maintaining data becomes easier as many different packages are created. All the variables must be private, and they should be handled by the only method in the class.

* + 1. **Which design pattern is based on encapsulation in Java?**

Ans:

Design pattern in encapsulation is a reusable solution for rectifying the problem occurring within the context. The design pattern is mainly used for creating layers of classes and modifying the changes becomes easier as a specific class is corrected/modified without affecting other classes.

* + 1. **Q: What is the difference between Encapsulation and Polymorphism?**

Ans:

Packaging data and methods in a single unit are Encapsulation. Their own methods are used for accessing the data which are hidden in the objects.

The word polymorphism means Poly means many and morph means form.

Polymorphism means the ability to take more than one form by using a single interface. when single or many types are not defined by name it represents any type by using abstract symbols. Polymorphism is a process used for implementing inherited data.

* 1. **Inheritance**

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

* + 1. **Why use inheritance in java?**

For Method Overriding (so runtime polymorphism can be achieved).

For Code Reusability.

Terms used in Inheritance

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

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

class Vehicle {

protected String brand = "Ford";

public void honk() {

System.out.println("Tuut, tuut!");

}

}

class Car extends Vehicle {

private String modelName = "Mustang";

public static void main(String[] args) {

Car myFastCar = new Car();

myFastCar.honk();

System.out.println(myFastCar.brand + " " + myFastCar.modelName);

}

}

* + 1. **What are different types of Inheritance supported by Java?**

Single Inheritance: One class is extended by only one class.

Multilevel Inheritance: One class is extended by a class and that class in turn is extended by another class thus forming a chain of inheritance.

Hierarchical Inheritance: One class is extended by many classes.

Hybrid Inheritance: It is a combination of above types of inheritance.

Multiple Inheritance: One class extends more than one classes. (Java does not support multiple inheritance.)

* + 1. **Can a class extend more than one classes or does java support multiple inheritance? If not, why?**

Java does not support multiple inheritance. This feature is avoided intentionally to avoid ambiguity, complexity and confusion.

For example, If Class C extends Class A and Class B which have a method with same name, then Class C will have two methods with same name.

This causes ambiguity and confusion for which method to use. To avoid this, java does not supports multiple inheritance.

class A

{

void methodOne()

{

System.out.println("From methodOfClassA");

}

}

class B

{

void methodOne()

{

System.out.println("From methodOfClassB");

}

}

class C extends A,B (If it is supported)

{

//two same methods will be inherited to Class C.

//This causes ambiguity and confusion.

}

* + 1. **What is Method Overriding And Method Hiding in Java?**

Method Overriding:

An instance method in a subclass with the same signature (name, number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method.

The overriding method has the same name, number and type of parameters, and return type as the method that it overrides.

An overriding method can also return a subtype of the type returned by the overridden method. This subtype is called a covariant return type.

When overriding a method, you might want to use the @Override annotation that instructs the compiler that you intend to override a method in the superclass.

If, for some reason, the compiler detects that the method does not exist in one of the superclasses, then it will generate an error.

Method Hiding:

If a subclass defines a static method with the same signature as a static method in the superclass, then the method in the subclass hides the one in the superclass.

The distinction between hiding a static method and overriding an instance method has important implications:The version of the overridden instance method that gets invoked is the one in the subclass.

The version of the hidden static method that gets invoked depends on whether it is invoked from the superclass or the subclass.

Consider an example that contains two classes. The first is Animal, which contains one instance method and one static method:

public class Animal {

public static void testClassMethod() {

System.out.println("The static method in Animal");

}

public void testInstanceMethod() {

System.out.println("The instance method in Animal");

}

}

The second class, a subclass of Animal, is called Cat:

public class Cat extends Animal {

public static void testClassMethod() {

System.out.println("The static method in Cat");

}

public void testInstanceMethod() {

System.out.println("The instance method in Cat");

}

public static void main(String[] args) {

Cat myCat = new Cat();

Animal myAnimal = myCat;

Animal.testClassMethod();

myAnimal.testInstanceMethod();

}

}

The Cat class overrides the instance method in Animal and hides the static method in Animal. The main method in this class creates an instance of Cat and invokes testClassMethod() on the class and testInstanceMethod() on the instance.

The output from this program is as follows:

The static method in Animal

The instance method in Cat

As promised, the version of the hidden static method that gets invoked is the one in the superclass, and the version of the overridden instance method that gets invoked is the one in the subclass.

Summary

The following table summarizes what happens when you define a method with the same signature as a method in a superclass.

Defining a Method with the Same Signature as a Superclass's Method

Superclass Instance Method Superclass Static Method

Subclass Instance Method Overrides Generates a compile-time error

Subclass Static Method Generates a compile-time error Hides

Note: In a subclass, you can overload the methods inherited from the superclass.

Such overloaded methods neither hide nor override the superclass instance methods—they are new methods, unique to the subclass.

* + 1. **Question: What is Method Overriding And Method Hiding in Java?**

Super class field will be hidden in the sub class. You can access hidden super class field in sub class using super keyword.

* + 1. **Question: A class member declared protected becomes member of subclass of which type?**

A class member declared protected becomes private member of subclass.

* + 1. **Question: How do you restrict a member of a class from inheriting to it’s sub classes?**

By declaring that member as a private. Because, private members are not inherited to sub classes.

* + 1. **Question: Are constructors and initializers also inherited to sub classes in Java?**

No, Constructors and initializers(Static initializers and instance initializers) are not inherited to sub classes.

But, they are executed while instantiating a sub class.

* + 1. **Question: Are static members inherited to sub classes in Java?**

Yes, Static members are also inherited to sub classes.

* + 1. **Question: What is Super Keyword In Java?**

The super keyword in java is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Super keyword has three purposes

super is used to refer immediate parent class instance variable.

super() is used to invoke immediate parent class constructor.

super is used to invoke immediate parent class method.

Question: What are some characteristics of constructors in Super and child classes in Java?

Most important points about constructors are

Constructors are not inherited.

If you do not make a constructor, the default empty constructor is automatically created.

If any constructor does not explicitly call a super or this constructor as its first statement, a call to super() is automatically added.

In Java, constructor of base class with no argument gets automatically called in derived class constructor. For example, output of following program is:

Base Class Constructor Called

Derived Class Constructor Called

// filename: Main.java

class Base {

Base() {

System.out.println("Base Class Constructor Called ");

}

}

class Derived extends Base {

Derived() {

System.out.println("Derived Class Constructor Called ");

}

}

public class Main {

public static void main(String[] args) {

Derived d = new Derived();

}

}

But, if we want to call parameterized contructor of base class, then we can call it using super().

The point to note is base class constructor call must be the first line in derived class constructor.

For example, in the following program, super(\_x) is first line derived class constructor.

// filename: Main.java

class Base {

int x;

Base(int \_x) {

x = \_x;

}

}

class Derived extends Base {

int y;

Derived(int \_x, int \_y) {

super(\_x);

y = \_y;

}

void Display() {

System.out.println("x = "+x+", y = "+y);

}

}

public class Main {

public static void main(String[] args) {

Derived d = new Derived(10, 20);

d.Display();

}

}

Output:

x = 10, y = 20

* + 1. **Question: What is Method Overloading In Java?**

Overloading is a process of declaring two methods with same name but different method signature

E.g. System.out which is object of PrintStream class has several println() method to print different data types e.g. byte, short, int, char, float and double.

All of them are called overloaded method. Overloaded method calls are resolved during compile time in Java and they must have different method signatures.

* + 1. **Question: What are rules of overloading a method in Java?**

The only rule of method overloading is that method signature of all overloaded method must be different.

Method signature is changed by changing either number of method arguments, or type of method arguments e.g. System.out.println() method is overloaded to accept different primitive types like int, short, byte, float etc.

They all accept just one argument but their type is different.

You can also change method signature by changing order of method argument but that often leads to ambiguous code so better to be avoided.

* + 1. **Can you prevent overriding a method without using final modifier?**

Yes, there are some funky ways to prevent method overriding in Java. Though final modifier is only for that purpose you can use private keyword to prevent method overriding.

How? If you remember correctly, in order to override a method, the class must be extensible. If you make the constructor of parent class private then its not possible to extend that class because its constructor will not be accessible in sub class.

Which is automatically invoked by sub class constructor, hence its not possible to override any method from that class.

This technique is used in Singleton design pattern, where constructor is purposefully made private and a static getInstance() method is provided to access singleton instance.

* + 1. **Question: Can We Override a Private Method in Java?**

No, you cannot override private method in Java. Since private methods are not visible outside the class, they are not available in sub-class hence they cannot be overridden.

* + 1. **What is co-variant Method Overriding?**

One of the rule of method overriding is that return type of overriding method must be same as overridden method but this restriction is relaxed little bit from Java 1.5 and now overridden method can return sub class of return type of original method.

This relaxation is known as co-variant method overriding and it allows you to remove casting at client end.

One of the best example of this comes when you override clone() method. Original Object.clone() method returns Object which needs to cast, but with co-variant method overriding you can directly return relevant type

E.g. Date class returns object of java.util.Date instead of java.lang.Object.

* + 1. **Can we change argument list of overridden method?**

No, you cannot change the argument list of overridden method in Java. An overriding method must have same signature as original method.

Only return type can be changed that to only to sub type of return type of original method.

* + 1. **Question: Can we change return type of method in subclass while overriding?**

No, you cannot change the return type of method during overriding. It would be violation of rules of overriding.

Though from Java 5 onward you can replace the return type with sub type e.g. if original method has return type as java.lang.Object then you can change return type of overridden method as any type e.g. clone() method.

This is also known as co-variant method overriding in Java.

* + 1. **Question: Can we override a method which throws run-time exception without throws clause?**

Yes, you can. There is no restriction on throwing RuntimeException from overriding method.

So if your original method throws NullPointerException than its not necessary to throw NPE from overriding method as well.

* + 1. **Question: How to call super class version of an overriding method in sub class?**

You can call it using super keyword. For example if you have a method draw() in both parent and child class.

Then from child class you can invoke parent class method draw() as super.draw(). It's very similar to calling super class constructor from sub class.

* + 1. **Question: What are rules of method overriding in Java?**

Some rules of method overriding are following:

Overriding method cannot throw higher exception than overridden one, but that's only true for checked exception.

Overriding method cannot restrict access of overridden method e.g. if original method is public then overriding method must be public.

But it can expand access e.g. if original method is protected than overriding method can be protected or public.

* + 1. **Question: Can we override a non-static method as static in Java?**

No, its not possible to define a non-static method of same name as static method in parent class, its compile time error in Java.

* + 1. **Question: Can we override constructor in Java?**

No, you cannot override constructor in Java because they are not inherited.

Remember, we are talking about overriding here not overloading, you can overload construct but you cannot override them.

Overriding always happens at child class and since constructor are not inherited and their name is always same as the class name its not possible to override them in Java.

* + 1. **Question: Can we override final method in Java?**

No, you cannot override final method in Java. Trying to override final method in subclass will result in compile time error.

Actually making a method final is signal to all developer that this method is not for inheritance and it should be use in its present form.

You generally make a method final due to security reasons.

* + 1. **Question: Can you overload or override main method in Java?**

Since main() is a static method in Java, it follows rules associated with static method, which means you can overload main method but you cannot override it.

By the way, even if you overload a main method, JVM will always call the standard public static void main(String args[]) method to start your program.

If you want to call your overloaded method you need to do it explicitly in your code.

* + 1. **Difference between method overloading and overriding?**

Fundamental difference between overloading and overriding is that former took place during compile time while later took place during run-time. Due to this reason.

Its only possible to overload virtual methods in Java. You cannot overload methods which are resolved during compile time e.g. private, static and final method cannot be overridden in Java.

Also, rules of method overloading and overriding are different, for example in order to overload a method its method signature must be different but for overriding method it must be same.

* + 1. **Super() vs This() in Java**

Super()

Super keyword is used to call constructor in the super class.

Super always refers to the parent of the current class

Super allows you to access public/protected method/attributes of parent class. You cannot see the parent's private method/attributes.

Super allows access to constructors from within the class' constructors only.

this()

this refers to a reference of the current class.

this allows access methods/attributes of the current class (including its own private methods/attributes).

this is used to access the methods and fields of the current object. For this reason, it has no meaning in static methods, for example. this keyword use to call constructor in the same class (other overloaded constructor)

* + 1. **Following code is showing compile time error. Can you identify the error?**

class X

{

//Class X Members

}

class Y

{

//Class Y Members

}

class Z extends X, Y

{

//Class Z Members

}

Answer:

In Java, a class can not extend more than one class. Class Z is extending two classes – Class X and Class Y. It is a compile time error in java.

* + 1. **Question: What will be the output of following program?**

class A

{

int i = 10;

}

class B extends A

{

int i = 20;

}

public class MainClass

{

public static void main(String[] args)

{

A a = new B();

System.out.println(a.i);

}

}

Answer:

10

* + 1. **Question: What will be the output of the below program?**

class A

{

{

System.out.println(1);

}

}

class B extends A

{

{

System.out.println(2);

}

}

class C extends B

{

{

System.out.println(3);

}

}

public class MainClass

{

public static void main(String[] args)

{

C c = new C();

}

}

Answer:

1 2 3

* + 1. **Question: How to identify super and invocation hierarchy ?**

class A

{

String s = "Class A";

}

class B extends A

{

String s = "Class B";

{

System.out.println(super.s);

}

}

class C extends B

{

String s = "Class C";

{

System.out.println(super.s);

}

}

public class MainClass

{

public static void main(String[] args)

{

C c = new C();

System.out.println(c.s);

}

}

Answer:

Class A

Class B

Class C

* + 1. **Question: What is the invocation flow in following example ?**

class A

{

static

{

System.out.println("THIRD");

}

}

class B extends A

{

static

{

System.out.println("SECOND");

}

}

class C extends B

{

static

{

System.out.println("FIRST");

}

}

public class MainClass

{

public static void main(String[] args)

{

C c = new C();

}

}

Answer:

THIRD SECOND FIRST

* + 1. **Question: Where super() or this() should be located?**

public class A

{

public A()

{

System.out.println(1);

super();

System.out.println(2);

Answer: Constructor calling statements ( super() or this() ), if written, must be the first statements in the constructor.

* + 1. **Question: Can you identify the member visibilty error in this code ?**

class X

{

private int m = 48;

}

class Y extends X

{

void methodOfY()

{

System.out.println(m);

}

}

Answer:

Because, private field ‘m’ is not visible to class Y.

* 1. **Abstraction**
     1. **What is Abstraction in Java?**

Ans:

Abstraction is a design concept where unwanted data is hidden from the user and result is derived for useful data. Abstraction is the design concept where only functionality is declared and does not define it. In other words, Abstraction only focuses on the object and not how it is implemented.

* + 1. **Q: How Abstraction is helpful in java?**

Ans:

Abstraction is one of the main concepts of OOPS(Object-oriented program). Abstraction is very much useful in Java as it reduces the complexity and efforts of programming from selecting Data from the big pool and showing results which is important for the users.

Abstraction only focuses on the object and not on its implementation. In simple words, Abstraction is a method of hiding certain details from end-users and only showing essential information which is necessary for users.

* + 1. **Q:What are the advantages of Abstraction?**

Ans:

The advantages of Abstraction are:

It is helpful in reducing the complexity of data.

It derives the result for a specific class.

It is helpful in hiding unwanted details from users.

The best example of Abstraction is Television. To use the Television, we do not need to understand how it works internally**.**

* + 1. **Q: How Abstraction is achieved in java?**

Abstraction can be achieved by declaring a specific keyword abstract. The keyword abstract is used as a non-access modifier with the class and method.

Abstraction can be accomplished through the use of abstract classes or interfaces. Abstraction can be used with methods and classes.

abstract with class:

abstract with the method:

Let understand in details :

abstract with class

Abstraction class means when the word abstract is used with class, then no one can instantiate that class. The classes and the methods of implication can be extended.

Important points to remember:

An abstract class is always declared with the keyword 'abstract'.

It can have both abstract, non-abstract methods.

It is non-instantiable.

It must have static and concrete methods.

The sub-class body and the method cannot be changed.

Example of abstract class:

abstract class RBI{

abstract int getRateOfInterest();

}

class CBN extends RBI{

int getRateOfInterest(){return 6;}

}

class KNB extends RBI{

int getRateOfInterest(){return 9;}

}

class TestRBI{

public static void main(String args[]){

RBI b;

b=new CBN();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

b=new KNB();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

}}

//Output

Rate of Interest is: 6 %

Rate of Interest is: 9

abstract with the method

Abstraction with the method means when the word abstract is used with the method where the overridden is must for the first concrete class. The implementation of this method is done where there is only an abstract class and the subclass which is provided that is inherited.

Important points to remember:

It must have a concrete class.

It does not have any implementation with the body.

It is used when the class is abstracted.

The implementation is achieved when the sub-class is inherited.

Example of abstract method:

abstract class Bird {

// Abstract method does not have a body

public abstract void birdSound();

// Common actions

public void sleep() {

System.out.println("Azzz");

}

}

// Subclass

class Parrot extends Bird {

public void birdSound() {

// The implementation of birdSound()

System.out.println("Theparrotsays: bzzz bzzz");

}

}

class Main {

public static void main(String[] args) {

Parrot myParrot = new Parrot();

myParrot.birdSound();

myParrot.sleep();

}

}

------------------------------------------

Output:

The Parrot says Azzz, Azzz,

bzzz.

* + 1. **Q:When to use abstract class in Java?**

Ans:

The abstract class is used when the class is defined by the keyword 'abstract'.

Following are the ways to use the abstract class:

The class must be written/assigned as an abstract class.

The hierarchy is maintained by using abstract class.

In abstract class types of behavior and implementation details can be known.

abstract class class\_name {

//abstract or non-abstract methods

}

* + 1. **Q:When to use the abstract method in Java?**

Ans:

Following are the ways to use the abstract method:

The abstract method is used when the word 'abstract' is used in abstract class.

The abstract method is used when the class is concrete.

It declares two more sub-classes.

The classes must be Overridden

* 1. **Polymorphism**

1. **Java basic constructs**
   1. **Data Types**
   2. **Loops**
2. **String Handling:**

Here are list of important method in the string class and their brief description. Please refer to java documentation for full list of methods.

* + - 1. [charAt](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#charAt-int-)(int index): Returns the char value at the specified index.
      2. [compareTo](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-)([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) anotherString):Compares two strings lexicographically(alphabetically)
      3. [compareToIgnoreCase](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareToIgnoreCase-java.lang.String-)([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) str)Compares two strings lexicographically, ignoring case differences.
      4. [concat](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#concat-java.lang.String-)([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) str): Concatenates the specified string to the end of this string.
      5. [contains](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#contains-java.lang.CharSequence-)([CharSequence](https://docs.oracle.com/javase/8/docs/api/java/lang/CharSequence.html) s):Returns true if and only if this string contains the specified sequence of char values.
      6. [getBytes](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#getBytes--)():Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
      7. [indexOf](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#indexOf-int-)(int ch):Returns the index within this string of the first occurrence of the specified character.
      8. [indexOf](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#indexOf-int-int-)(int ch, int fromIndex):Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
      9. [indexOf](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#indexOf-java.lang.String-int-)([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) str, int fromIndex):Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
      10. [isEmpty](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#isEmpty--)(): Returns true if, and only if, [length()](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#length--) is 0.
      11. [length](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#length--)():Returns the length of this string
      12. [split](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#split-java.lang.String-)([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) regex):Splits this string around matches of the given [regular expression](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#sum).
      13. [split](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#split-java.lang.String-int-)([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) regex, int limit): Splits this string around matches of the given [regular expression](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#sum).
      14. [substring](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#substring-int-)(int beginIndex):Returns a string that is a substring of this string.
      15. [substring](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#substring-int-int-)(int beginIndex, int endIndex):Returns a string that is a substring of this string.
      16. [toCharArray](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#toCharArray--)(): Converts this string to a new character array.
      17. [toLowerCase](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#toLowerCase--)(): Converts all of the characters in this String to lower case using the rules of the default locale .
      18. [toUpperCase](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#toUpperCase--)():Converts all of the characters in this String to upper case using the rules of the default locale.
      19. [trim](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#trim--)(): Returns a string whose value is this string, with any leading and trailing whitespace removed.
    1. **what is string in java ?**
       1. Strings, one of the most common objects used in Java programming, are essentially sequences of characters. As an example, the string "Scaler" contains the following characters: "S", "c", "a", "l", "e", and "r". You can either create a string by using String Literal or by using the NEW keyword. Additionally, String supports a variety of methods to operate on Strings, such as the equals method to compare two Strings, the replace method to replace String characters, the substring method to get a substring, the toUpperCase method to convert String to upper case, the split method to split a long String into multiple Strings, and so on.
       2. String is a Class in java and defined in java.lang package. It’s not a primitive data type like int and long. String class represents character Strings. String is used in almost all the Java applications and there are some interesting facts we should know about String. String in immutable and final in Java and JVM uses String Pool to store all the String objects. Some other interesting
       3. things about String is the way we can instantiate a String object using double quotes and overloading of “+” operator for concatenation.
    2. **what are different ways to create string objects ?**
       1. We can create String object using new operator like any normal java class or we can use double quotes to create a String object. There are several constructors available in String class to get String from char array, byte array, StringBuffer and StringBuilder.
       2. String str = new String("abc");
       3. String str1 = "abc";
       4. When we create a String using double quotes, JVM looks in the String pool to find if any other String is stored with the same value. If found, it just returns the reference to that String object else it creates a new String object with given value and stores it in the String pool. When we use the new operator, JVM creates the String object but don’t store it into the String Pool. We can use intern() method to store the String object into String pool or return the reference if there is already a String with equal value present in the pool.
    3. **Write a method to check if input String is Palindrome?**
       1. A String is said to be Palindrome if it’s value is same when reversed. For example “aba” is a Palindrome String. String class doesn’t provide any method to reverse the String but StringBuffer and StringBuilder class has reverse method that we can use to check if String is palindrome or not.
       2. private static boolean isPalindrome(String str) {
          1. if (str == null)
          2. return false;
          3. StringBuilder strBuilder = new StringBuilder(str);
          4. strBuilder.reverse();
          5. return strBuilder.toString().equals(str);
       3. }
       4. Sometimes interviewer asks not to use any other class to check this, in that case, we can compare characters in the String from both ends to find out if it’s palindrome or not.
       5. private static boolean isPalindromeString(String str) {

if (str == null)

return false;

int length = str.length();

System.out.println(length / 2);

for (int i = 0; i < length / 2; i++) {

if (str.charAt(i) != str.charAt(length - i - 1))

return false;

}

return true;

}

* + 1. **Write a method that will remove given character from the String?**
       1. We can use replaceAll method to replace all the occurance of a String with another String. The important point to note is that it accepts String as argument, so we will use Character class to create String and use it to replace all the characters with empty String.
       2. private static String removeChar(String str, char c) {
          1. if (str == null)
          2. return null;
          3. return str.replaceAll(Character.toString(c), "");
       3. }
    2. **How can we make String upper case or lower case?**
       1. We can use String class toUpperCase and toLowerCase methods to get the String in all upper case or lower case. These methods have a variant that accepts Locale argument and use that locale rules to convert String to upper or lower case.
    3. **What is String subSequence method?**
       1. Java 1.4 introduced CharSequence interface and String implements this interface, this is the only reason for the implementation of subSequence method in String class. Internally it invokes the String substring method. Check this post for String subSequence example.
    4. **How to compare two Strings in java program?**
       1. Java String implements Comparable interface and it has two variants of compareTo() methods. compareTo(String anotherString) method compares the String object with the String argument passed lexicographically. If String object precedes the argument passed, it returns negative integer and if String object follows the argument String passed, it returns a positive integer. It returns zero when both the String have the same value, in this case equals(String str) method will also return true. compareToIgnoreCase(String str): This method is similar to the first one, except that it ignores the case. It uses String CASE\_INSENSITIVE\_ORDER Comparator for case insensitive comparison. If the value is zero then equalsIgnoreCase(String str) will also return true. Check this post for String compareTo example.
       2. How to convert String to char and vice versa?
       3. This is a tricky question because String is a sequence of characters, so we can’t convert it to a single character. We can use use charAt method to get the character at given index or we can use toCharArray() method to convert String to character array. Check this post for sample program on converting String to character array to String.
    5. **How to convert String to byte array and vice versa?**
       1. We can use String getBytes() method to convert String to byte array and we can use String constructor new String(byte[] arr) to convert byte array to String. Check this post for String to byte array example.
    6. **Can we use String in switch case?**
       1. This is a tricky question used to check your knowledge of current Java developments. Java 7 extended the capability of switch case to use Strings also, earlier Java versions don’t support this. If you are implementing conditional flow for Strings, you can use if-else conditions and you can use switch case if you are using Java 7 or higher versions. Check this post for Java Switch Case String example.
    7. **Write a program to print all permutations of String?**
       1. This is a tricky question and we need to use recursion to find all the permutations of a String, for example “AAB” permutations will be “AAB”, “ABA” and “BAA”. We also need to use Set to make sure there are no duplicate values. Check this post for complete program to find all permutations of String.
    8. **Write a function to find out longest palindrome in a given string?**
       1. A String can contain palindrome strings in it and to find longest palindrome in given String is a programming question. Check this post for complete program to find longest palindrome in a String.
    9. **Difference between String, StringBuffer and StringBuilder?**
       1. The string is immutable and final in Java, so whenever we do String manipulation, it creates a new String. String manipulations are resource consuming, so java provides two utility classes for String manipulations - StringBuffer and StringBuilder. StringBuffer and StringBuilder are mutable classes. StringBuffer operations are thread-safe and synchronized where StringBuilder operations are not thread-safe. So in a multi-threaded environment, we should use StringBuffer but in the single-threaded environment, we should use StringBuilder. StringBuilder performance is fast than StringBuffer because of no overhead of synchronization. Check this post for extensive details about String vs StringBuffer vs StringBuilder. Read this post for benchmarking of StringBuffer vs StringBuilder.
    10. **Why String is immutable or final in Java**
        1. There are several benefits of String because it’s immutable and final.
        2. String Pool is possible because String is immutable in java.
        3. It increases security because any hacker can’t change its value and it’s used for storing sensitive information such as database username, password etc.
        4. Since String is immutable, it’s safe to use in multi-threading and we don’t need any synchronization.
        5. Strings are used in java classloader and immutability provides security that correct class is getting loaded by Classloader.
    11. **How to Split String in java?**
        1. We can use split(String regex) to split the String into String array based on the provided regular expression. Learn more at java String split.
    12. **Why Char array is preferred over String for storing password?**
        1. String is immutable in Java and stored in String pool. Once it’s created it stays in the pool until unless garbage collected, so even though we are done with password it’s available in memory for longer duration and there is no way to avoid it. It’s a security risk because anyone having access to memory dump can find the password as clear text. If we use a char array to store password, we can set it to blank once we are done with it. So we can control for how long it’s available in memory that avoids the security threat with String.
    13. **How do you check if two Strings are equal in Java?**
        1. There are two ways to check if two Strings are equal or not - using “==” operator or using equals method. When we use “==” operator, it checks for the value of String as well as the reference but in our programming, most of the time we are checking equality of String for value only. So we should use the equals method to check if two Strings are equal or not. There is another function equalsIgnoreCase that we can use to ignore case.

String s1 = "abc";

String s2 = "abc";

String s3= new String("abc");

System.out.println("s1 == s2 ? "+(s1==s2)); //true

System.out.println("s1 == s3 ? "+(s1==s3)); //false

System.out.println("s1 equals s3 ? "+(s1.equals(s3))); //true

* + 1. **What is String Pool?**
       1. As the name suggests, String Pool is a pool of Strings stored in Java heap memory. We know that String is a special class in Java and we can create String object using new operator as well as providing values in double quotes. Check this post for more details about String Pool.
    2. **What does String intern() method do?**
       1. When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned. This method always returns a String that has the same contents as this string but is guaranteed to be from a pool of unique strings.
    3. **Does String is thread-safe in Java?**
       1. Strings are immutable, so we can’t change it’s value in program. Hence it’s thread-safe and can be safely used in multi-threaded environment. Check this post for Thread Safety in Java.
    4. **Why String is popular HashMap key in Java?**
       1. Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for the key in a Map and it’s processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.
    5. String Programming Questions
    6. **What is the output of below program?**

package com.journaldev.strings;

public class StringTest {

public static void main(String[] args) {

String s1 = new String("java");

String s2 = new String("PYTHON");

System.out.println(s1 = s2);

}

}

It’s a simple yet tricky program, it will print “PYTHON” because we are assigning s2 String to s1. Don’t get confused with == comparison operator.

* + 1. **What is the output of below program?**

package com.journaldev.strings;

public class Test {

public void foo(String s) {

System.out.println("String");

}

public void foo(StringBuffer sb){

System.out.println("StringBuffer");

}

public static void main(String[] args) {

new Test().foo(null);

}

}

The above program will not compile with error as “The method foo(String) is ambiguous for the type Test”. For complete clarification read Understanding the method X is ambiguous for the type Y error.

* + 1. **What is the output of below code snippet?**

String s1 = new String("abc");

String s2 = new String("abc");

System.out.println(s1 == s2);

It will print false because we are using new operator to create String, so it will be created in the heap memory and both s1, s2 will have different reference. If we create them using double quotes, then they will be part of string pool and it will print true.

* + 1. **What will be output of below code snippet?**

String s1 = "abc";

StringBuffer s2 = new StringBuffer(s1);

System.out.println(s1.equals(s2));

It will print false because s2 is not of type String. If you will look at the equals method implementation in the String class, you will find a check using instanceof operator to check if the type of passed object is String? If not, then return false.

* + 1. **What will be the output of below program?**

String s1 = "abc";

String s2 = new String("abc");

s2.intern();

System.out.println(s1 ==s2);

It’s a tricky question and output will be false. We know that intern() method will return the String object reference from the string pool, but since we didn’t assigned it back to s2, there is no change in s2 and hence both s1 and s2 are having different reference. If we change the code in line 3 to s2 = s2.intern(); then output will be true.

* + 1. **How many String objects got created in below code snippet?**

String s1 = new String("Hello");

String s2 = new String("Hello");

The answer is 3. First - line 1, “Hello” object in the string pool. Second - line 1, new String with value “Hello” in the heap memory. Third - line 2, new String with value “Hello” in the heap memory. Here “Hello” string from string pool is reused.

1. **Exception Handling**
2. **Multithreading**
3. **Collection Framework**
4. **JDBC**
5. **Servlet**
6. **JSP**

**References:**

* + - 1. [**https://docs.oracle.com/en/java/**](https://docs.oracle.com/en/java/)
      2. [**https://www.java67.com/**](https://www.java67.com/2018/06/top-35-java-string-interview-questions.html)
      3. [**https://www.baeldung.com**](https://www.baeldung.com)
      4. **https://javahungry.blogspot.com/**
      5. [**https://www.javatpoint.com/**](https://www.javatpoint.com/)
      6. [**https://www.digitalocean.com/**](https://www.digitalocean.com/)
      7. [**www.interviewbit.com**](http://www.interviewbit.com)
      8. [**https://www.w3schools.com/**](https://www.w3schools.com/)
      9. [**http://www.codespaghetti.com/**](http://www.codespaghetti.com/)
      10. **https://www.techgeeknext.com/**