dsid- US7404210

33564085

35457493

35457493

[2/18 4:18 PM] mounika, sharab

smounika@kpmg.com

​

[2/18 4:19 PM] mounika, sharab

dsid- US7404210

Interface:

Interface is same like class but it has methods with only signature without body

All methods are abstract methods

Abstraction:

Abstraction is the process of hiding the implementations details from user, only the functionality will be provided to the user. In other words, user will have the information on what object does but not how it does it.

**Abstract class:**

\*) Class which have both abstract method (without implementation) and concrete methods (which has body) is called abstract class

\*) we cannot create object for abstract class

\*) we cannot give Private (as access modifier) to method in abstract class as main purpose is to inherit and use those methods in parent class

**Packages:** set of classes, interfaces...

Java.lang is a package where int,array all these are present but we are not importing them because this is default package that comes with java complier

Java.util – all collections interfaces comes from this package

Java.io – package for file input and output or file reading

Syntax to import a package

import packagename.classname

class A can use class b methods by directly creating obj for class b in class A if they both belong to same package if they are not from same package they need to import package

**Access modifiers:**

1)Default: if you are mentioned any access modifier it will take as default

Default: access methods or variables anywhere only in package

2) public: access methods or variables anywhere across packages

3) private: you cannot access variables or methods outside of class of same package

4)protected: variables/methods can be accessible only across subclasses (i.e inherited classes). It can be within or different package only thing is that class needs inherit that class

**Collections:**

Java collections framework is a collection of interfaces and classes which helps in storing and processing the data more efficiently. this framework has several useful classes which has tons of useful functions which makes programmer task super easy.

1)List: A list is an ordered collection (sometimes called as sequence). List may contain duplicate elements. Below are classes implement list interface

a) Array list

b) linked list

c)vector

2)Set: A set is a collection that cannot contain duplicate elements. However it makes no guarantee the concerning the order of iteration

Below are classes implement set interface

a) Hashset

b) linkedhashset

c) treeset

3)Map: A map is an object that maps keys to values. A map cannot contain duplicate keys.

Below are main implementation of map interfaces

a)Hashmap

b)treemap

c)linked hashmap

1. What are the core concepts of OOPS?

OOPS core concepts are;

* 1. Abstraction
  2. Encapsulation
  3. Polymorphism
  4. Inheritance
  5. Composition
  6. Association
  7. Aggregation

1. What is Abstraction?

Abstraction is an OOPS concept to construct the structure of the real world objects. During this construction only the general states and behaviors are taken and more specific states and behaviors are left aside for the implementers.

1. What is Encapsulation?

Encapsulation is an OOPS concept to create and define the permissions and restrictions of an object and its member variables and methods. A very simple example to explain the concept is to make the member variables of a class private and providing public getter and setter methods. Java provides four types of access level modifiers: public, protected, no modifier and private.

1. What is the difference between Abstraction and Encapsulation?
   1. “Program to interfaces, not implementations” is the principle for Abstraction and “Encapsulate what varies” is the OO principle for Encapsulation.
   2. Abstraction provides a general structure of a class and leaves the details for the implementers. Encapsulation is to create and define the permissions and restrictions of an object and its member variables and methods.
   3. Abstraction is implemented in Java using interface and abstract class while Encapsulation is implemented using four types of access level modifiers: public, protected, no modifier and private.
2. What is Polymorphism?

Polymorphism is the occurrence of something in various forms. Java supports various forms of polymorphism like polymorphic reference variables, polymorphic method, polymorphic return types and polymorphic argument types.

1. What is Inheritance?

A subclass can inherit the states and behaviors of it’s super class is known as inheritance.

1. What is multiple inheritance?

A child class inheriting states and behaviors from multiple parent classes is known as multiple inheritance.

1. What is the diamond problem in inheritance?

In case of multiple inheritance, suppose class A has two subclasses B and C, and a class D has two super classes B and C.If a method present in A is overridden by both B and C but not by D then from which class D will inherit that method B or C? This problem is known as diamond problem.

1. Why Java does not support multiple inheritance?

Java was designed to be a simple language and multiple inheritance introduces complexities like diamond problem. Inheriting states or behaviors from two different type of classes is a case which in reality very rare and it can be achieved easily through an object association.

1. What is Static Binding and Dynamic Binding?

Static or early binding is resolved at compile time. Method overloading is an example of static binding.

Dynamic or late or virtual binding is resolved at run time. Method overriding is an example of dynamic binding.

1. What is a Class?

A class is the specification or template of an object.

1. What is an Object?

Object is instance of class.

Why string is immutable

# **Immutable String in Java**

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

1. **class** Testimmutablestring{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s.concat(" Tendulkar");//concat() method appends the string at the end
5. System.out.println(s);//will print Sachin because strings are immutable objects
6. }
7. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=Testimmutablestring)

Output:Sachin

1. **class** Testimmutablestring1{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s=s.concat(" Tendulkar");
5. System.out.println(s);
6. }
7. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=Testimmutablestring1)

Output:Sachin Tendulkar

# **Difference between StringBuffer and StringBuilder**

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable. There are many differences between StringBuffer and StringBuilder. The StringBuilder class is introduced since JDK 1.5.

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe.  It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |

A list of differences between StringBuffer and StringBuilder are given below:

**String builder**

**public** **class** BuilderTest{

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

        StringBuilder builder=**new** StringBuilder("hello");

        builder.append("java");

        System.out.println(builder);

    }

}

String Buffer

        StringBuffer sb=**new** StringBuffer("java");

        System.out.println(sb.hashCode());

        sb.append("tpoint");

        System.out.println(sb.hashCode());

# What are wrapper classes in Java?

https://www.educative.io/cdn-cgi/image/f=auto,fit=cover,w=32,h=32/v2api/author/profile/6547532599525376/image/5933301341618176Edpresso Team

A **wrapper class** in Java converts a primitive data type into class object.

Java is an object-oriented language that only supports pass by value. Therefore, wrapper class objects allow us to change the original passed value. These wrapper classes help with multithreading and synchronization because, in Java, multithreading only works with objects. Moreover, most collections (like Vector, LinkedList, etc.) work with objects.

| **Primitve Data Type** | **Wrapper Class** |
| --- | --- |
| int | Integer |
| short | Short |
| byte | Byte |
| char | Character |
| long | Long |
| float | Float |
| double | Double |
| boolean | Boolean |

### Autoboxing

The automatic conversion of primitive data types into its wrapper class objects is known as autoboxing.

class Autoboxing {  
    public static void main( String args[] ) {  
     int a = 15; // Primitive data type  
     Integer I = a; // Autoboxing will occur internally.  
    }  
}

### Unboxing

The automatic conversion of wrapper class objects into its primitive data types is known as unboxing.

class Autoboxing {  
    public static void main( String args[] ) {  
      Integer a = new Integer(15); // Wrapper class object  
      int I = a;// Unboxing will occur internally.  
    }  
}

call by value

|  |
| --- |
| If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method. |
|  |

### **Example of call by value in java**

|  |
| --- |
| In case of call by value original value is not changed. Let's take a simple example: |

**class** Operation{

**int** data=50;

**void** change(**int** data){

 data=data+100;//changes will be in the local variable only

 }

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

   Operation op=**new** Operation();

   System.out.println("before change "+op.data);

   op.change(500);

   System.out.println("after change "+op.data);

 }

}

[download this example](https://www.javatpoint.com/src/oops/callbyvalue1.zip)

Output:before change 50

after change 50

### **Another Example of call by value in java**

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value. Let's take a simple example:

**class** Operation2{

**int** data=50;

**void** change(Operation2 op){

 op.data=op.data+100;//changes will be in the instance variable

 }

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

   Operation2 op=**new** Operation2();

   System.out.println("before change "+op.data);

   op.change(op);//passing object

   System.out.println("after change "+op.data);

 }

}

[download this example](https://www.javatpoint.com/src/oops/callbyvalue2.zip)

Output:before change 50

after change 150

# Difference between concat() and + operator in Java

[concat() Method](https://www.geeksforgeeks.org/java-string-concat-examples/)

The [Java String](https://www.geeksforgeeks.org/strings-in-java/) concat() method concatenates one string to the end of another string. This method returns a string with the value of the string passed into the method, appended to the end of the string.

+ operator is used to concatenate strings on either side.

| **Points** | **concat() method** | **+ operator** |
| --- | --- | --- |
| **Definition** | A **concat() method** is method to combine two strings . | **+ operator** used to concatenate any number of strings. |
| **Number of arguments** | In **concat()** method, takes only one argument of string and concatenate it with another string. | In **+ operator**takes any number of arguments and combines all strings. |
| **Type of arguments** | **concat()** method takes arguments of string type only. | **+ operator** takes any type of argument and converts it to string type and then combine them. |
| **Creates new string** | **concat()** takes concatenates two strings and return new string object only string length is greater than 0, otherwise it returns same object.. | **+ operator**creates a new string object every time irrespective of length of string. |
| **NullPointer Exception** | In **concat()** method raises NullPointer Exception when string is concatenated with null . | **+ operator** concatenates string with without any error. |
| **Performance** | **concat() method**is better than + operator because it creates a new object only when the string length is greater than zero(0), so it uses less amount of memory. | **+ operator**always a creates a new string irrespective of length of string therefore it takes more memory. |