\

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Basic Java Interview Questions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q1. Explain JVM, JRE and JDK?

JVM (Java Virtual Machine): It is an abstract machine. It is a specification that provides run-time environment in which java bytecode can be executed. It follows three notations:

Specification: It is a document that describes the implementation of the Java virtual machine. It is provided by Sun and other companies.

Implementation: It is a program that meets the requirements of JVM specification.

Runtime Instance: An instance of JVM is created whenever you write a java command on the command prompt and run the class.

JRE (Java Runtime Environment) : JRE refers to a runtime environment in which java bytecode can be executed. It implements the JVM (Java Virtual Machine) and provides all the class libraries and other support files that JVM uses at runtime. So JRE is a software package that contains what is required to run a Java program. Basically, it’s an implementation of the JVM which physically exists.

JDK(Java Development Kit) : It is the tool necessary to compile, document and package Java programs. The JDK completely includes JRE which contains tools for Java programmers. The Java Development Kit is provided free of charge. Along with JRE, it includes an interpreter/loader, a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development. In short, it contains JRE + development tools.

Refer to this below image and understand how exactly these components reside:

Components - Java Interview Questions - Edureka

Q2. Explain public static void main(String args[]).

public : Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.

static : It is a keyword in java which identifies it is class based i.e it can be accessed without creating the instance of a Class.

void : It is the return type of the method. Void defines the method which will not return any value.

main: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.

String args[] : It is the parameter passed to the main method.

Q3. Why Java is platform independent?

Platform independent practically means “write once run anywhere”. Java is called so because of its byte codes which can run on any system irrespective of its underlying operating system.

Q4. Why java is not 100% Object-oriented?

Java is not 100% Object-oriented because it makes use of eight primitive datatypes such as boolean, byte, char, int, float, double, long, short which are not objects.

Q5. What are wrapper classes?

Wrapper classes converts the java primitives into the reference types (objects). Every primitive data type has a class dedicated to it. These are known as wrapper classes because they “wrap” the primitive data type into an object of that class. Refer to the below image which displays different primitive type, wrapper class and constructor argument.

WrapperClass - Java Interview Questions - Edureka

Q6. What are constructors in Java?

In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.

There are two types of constructors:

Default constructor

Parameterized constructor

Q7. What is singleton class and how can we make a class singleton?

Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

Q8. What is the difference between Array list and vector?

Array List Vector

Array List is not synchronized. Vector is synchronized.

Array List is fast as it’s non-synchronized. Vector is slow as it is thread safe.

If an element is inserted into the Array List, it increases its Array size by 50%. Vector defaults to doubling size of its array.

Array List does not define the increment size. Vector defines the increment size.

Array List can only use Iterator for traversing an Array List. Except Hashtable, Vector is the only other class which uses both Enumeration and Iterator.

Q9. What is the difference between equals() and == ?

Equals() method is defined in Object class in Java and used for checking equality of two objects defined by business logic.

“==” or equality operator in Java is a binary operator provided by Java programming language and used to compare primitives and objects. public boolean equals(Object o) is the method provided by the Object class. The default implementation uses == operator to compare two objects. For example: method can be overridden like String class. equals() method is used to compare the values of two objects.

public class Equaltest {

public static void main(String[] args) {

String str1= new String(“ABCD”);

String str2= new String(“ABCD”);

if(Str1 == str2)

{

System.out.println("String 1 == String 2 is true");

}

else

{

System.out.println("String 1 == String 2 is false");

String Str3 = Str2;

if( Str2 == Str3)

{

System.out.println("String 2 == String 3 is true");

}

else

{

System.out.println("String 2 == String 3 is false");

}

if(Str1.equals(str2))

{

System.out.println("String 1 equals string 2 is true");

}

else

{

System.out.prinltn("String 1 equals string 2 is false");

}

}}

Q10. What are the differences between Heap and Stack Memory?

The major difference between Heap and Stack memory are:

Features Stack Heap

Memory Stack memory is used only by one thread of execution.

Heap memory is used by all the parts of the application.

Access Stack memory can’t be accessed by other threads.

Objects stored in the heap are globally accessible.

Memory Management

Follows LIFO manner to free memory.

Memory management is based on generation associated to each object.

Lifetime Exists until the end of execution of the thread.

Heap memory lives from the start till the end of application execution.

UsageStack memory only contains local primitive and reference variables to objects in heap space.

Whenever an object is created, it’s always stored in the Heap space.

Polymorphism is briefly described as “one interface, many implementations”. Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts – specifically, to allow an entity such as a variable, a function, or an object to have more than one form. There are two types of polymorphism:

Compile time polymorphism

Run time polymorphism

Compile time polymorphism is method overloading whereas Runtime time polymorphism is done using inheritance and interface.

Q2. What is runtime polymorphism or dynamic method dispatch?

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. Let’s take a look at the example below to understand it better.

class Car {

void run()

{

System.out.println(“car is running”);

}

}

class Audi extends Car {

void run()

{

System.out.prinltn(“Audi is running safely with 100km”);

}

public static void main(String args[])

{

Car b= new Audi(); //upcasting

b.run();

}

}

Q3. What is the difference between abstract classes and interfaces?

Abstract Class Interfaces

An abstract class can provide complete, default code and/or just the details that have to be overridden. An interface cannot provide any code at all,just the signature.

In case of abstract class, a class may extend only one abstract class. A Class may implement several interfaces.

An abstract class can have non-abstract methods. All methods of an Interface are abstract.

An abstract class can have instance variables. An Interface cannot have instance variables

An abstract class can have any visibility: public, private, protected. An Interface visibility must be public (or) none.

If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method

An abstract class can contain constructors An Interface cannot contain constructors

Abstract classes are fast Interfaces are slow as it requires extra indirection to find corresponding method in the actual class

Q4. What is method overloading and method overriding?

Method Overloading :

In Method Overloading, Methods of the same class shares the same name but each method must have different number of parameters or parameters having different types and order.

Method Overloading is to “add” or “extend” more to method’s behavior.

It is a compile time polymorphism.

The methods must have different signature.

It may or may not need inheritance in Method Overloading.

Let’s take a look at the example below to understand it better.

class Adder {

Static int add(int a, int b)

{

return a+b;

}

Static double add( double a, double b)

{

return a+b;

}

public static void main(String args[])

{

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(12.3,12.6));

}}

Method Overriding:

In Method Overriding, sub class have the same method with same name and exactly the same number and type of parameters and same return type as a super class.

Method Overriding is to “Change” existing behavior of method.

It is a run time polymorphism.

The methods must have same signature.

It always requires inheritance in Method Overriding.

Let’s take a look at the example below to understand it better.

class Car {

void run(){

System.out.println(“car is running”);

}

Class Audi extends Car{

void run()

{

System.out.prinltn(“Audi is running safely with 100km”);

}

public static void main( String args[])

{

Car b=new Audi();

b.run();

}

}

Q5. Can you override a private or static method in Java?

You cannot override a private or static method in Java. If you create a similar method with same return type and same method arguments in child class then it will hide the super class method; this is known as method hiding. Similarly, you cannot override a private method in sub class because it’s not accessible there. What you can do is create another private method with the same name in the child class. Let’s take a look at the example below to understand it better.

class Base {

private static void display() {

System.out.println("Static or class method from Base");

}

public void print() {

System.out.println("Non-static or instance method from Base");

}

class Derived extends Base {

private static void display() {

System.out.println("Static or class method from Derived");

}

public void print() {

System.out.println("Non-static or instance method from Derived");

}

public class test {

public static void main(String args[])

{

Base obj= new Derived();

obj1.display();

obj1.print();

}

}

Q6. What is multiple inheritance? Is it supported by Java?

MultipleInheritance - Java Interview Questions - EdurekaIf a child class inherits the property from multiple classes is known as multiple inheritance. Java does not allow to extend multiple classes.

The problem with multiple inheritance is that if multiple parent classes have a same method name, then at runtime it becomes difficult for the compiler to decide which method to execute from the child class.

Therefore, Java doesn’t support multiple inheritance. The problem is commonly referred as Diamond Problem.

Q7. What is association?

Association is a relationship where all object have their own lifecycle and there is no owner. Let’s take an example of Teacher and Student. Multiple students can associate with a single teacher and a single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. These relationship can be one to one, One to many, many to one and many to many.

Q8. What do you mean by aggregation?

Aggregation is a specialized form of Association where all object have their own lifecycle but there is ownership and child object can not belongs to another parent object. Let’s take an example of Department and teacher. A single teacher can not belongs to multiple departments, but if we delete the department teacher object will not destroy.

Q9. What is composition in Java?

Composition is again specialized form of Aggregation and we can call this as a “death” relationship. It is a strong type of Aggregation. Child object dose not have their lifecycle and if parent object deletes all child object will also be deleted. Let’s take again an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belongs to two different house if we delete the house room will automatically delete.

Explain the creation of a thread-safe singleton in Java using double-checks locking

Singleton is created with double checked locking as before Java 5 acts as an broker and it’s been possible to have multiple instances of Singleton when multiple threads creates an instance of Singleton at the same time. Java 5 made it easy to create thread-safe Singleton using Enum. Using a volatile variable is essential for the same.

Differentiate between StringBuffer and StringBuilder in Java programming.

String Buffer String Builder

StringBuffer methods are synchronized StringBuilder is non synchronized

Storage area is Heap and modified easily. Storage is Heap and can be modified.

StringBuffer is thread safe. StringBuilder is fast as it is not thread safe

Performance is very slow Performance is very fast.

Java doesn't support multiple inheritance. Why?

Java doesn’t support multiple inheritance. Because we cannot use different methods in one class it creates an ambiguity.

Example:

class Intellipaat1

{

void test()

{

system.out.println("test() method");

}

}class Intellipaat2

{

void test()

{

system.out.println("test() method");

}

}Multiple inheritance

class C extends Intellipaat1, Intellipaat2

{

………………………………………….

…………………………………………..

}

Intellipaat1 and Intellipaat2 test() methods are inheriting to class C

So which test() method C class will take. As Intellipaat1 & Intellipaat2 class test () methods are different, So here we would face ambiguity.

18. Are constructors inherited? Can a subclass call the parent's class constructor?

We cannot inherit a constructor. We create an instance of a subclass using a constructor of one of its superclass. Because override the superclass constructor is not our wish so that, we override a superclass constructor, then we destroy the encapsulation abilities of the language.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

COLLECTION IN JAVA MIMP TOPIC

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

What is Collection ? What is a Collections Framework ? What are the benefits of Java Collections Framework ?

Collection : A collection (also called as container) is an object that groups multiple elements into a single unit.

Collections Framework : Collections framework provides unified architecture for manipulating and representing collections.

Benefits of Collections Framework :

1. Improves program quality and speed

2. Increases the chances of reusability of software

3. Decreases programming effort.

Q2 What is the root interface in collection hierarchy ?

Root interface in collection hierarchy is Collection interface . Few interviewer may argue that

Collection interface extends Iterable interface. So iterable should be the root interface. But you should reply iterable interface present in java.lang package not in java.util package .It is clearly mentioned in Oracle Collection docs , that Collection interface is a member of the Java Collections framework. For Iterable interface Oracle doc , iterable interface is not mentioned as a part of the Java Collections framework .So if the question includes collection hierarchy , then you should answer the question as Collection interface (which is found in java.util package).

Q3 What is the difference between Collection and Collections ?

Collection is an interface while Collections is a java class , both are present in java.util package and part of java collections framework.

Q4 Which collection classes are synchronized or thread-safe ?

Stack, Properties , Vector and Hashtable can be used in multi threaded environment because they are synchronized classes (or thread-safe).

6 What is the difference between List and Set ?

Set contain only unique elements while List can contain duplicate elements.

Set is unordered while List is ordered . List maintains the order in which the objects are added .

Q7 What is the difference between Map and Set ?

Map object has unique keys each containing some value, while Set contain only unique values.

Q8 What are the classes implementing List and Set interface ?

Class implementing List interface : ArrayList , Vector , LinkedList ,

Class implementing Set interface : HashSet , TreeSet

Q9 What is an iterator ?

Iterator is an interface . It is found in java.util package. It provides methods to iterate over any Collection.

Q10 What is the difference between Iterator and Enumeration ?

The main difference between Iterator and Enumeration is that Iterator has remove() method while Enumeration doesn't.

Hence , using Iterator we can manipulate objects by adding and removing the objects from the collections. Enumeration behaves like a read only interface as it can only traverse the objects and fetch it .

Q11 Which design pattern followed by Iterator ?

It follows iterator design pattern. Iterator design pattern provides us to navigate through the collection of objects by using a common interface without letting us know about the underlying implementation.

Enumeration is an example of Iterator design pattern.

Q12 Which methods you need to override to use any object as key in HashMap ?

To use any object as key in HashMap , it needs to implement equals() and hashCode() method .

Q13 What is the difference between Queue and Stack ?

Queue is a data structure which is based on FIFO ( first in first out ) property . An example of Queue in real world is buying movie tickets in the multiplex or cinema theaters.

Stack is a data structure which is based on LIFO (last in first out) property . An example of Stack in real world is insertion or removal of CD from the CD case.

Q14 How to reverse the List in Collections ?

There is a built in reverse method in Collections class . reverse(List list) accepts list as parameter.

Collections.reverse(listobject);

Q15 How to convert the array of strings into the list ?

Arrays class of java.util package contains the method asList() which accepts the array as parameter.

So,

String[] wordArray = {"Love Yourself" , "Alive is Awesome" , "Be in present"};

List wordList = Arrays.asList(wordArray);

16 What is the difference between ArrayList and Vector ?

It is one of the frequently asked collection interview question , the main differences are

Vector is synchronized while ArrayList is not . Vector is slow while ArrayList is fast . Every time when needed, Vector increases the capacity twice of its initial size while ArrayList increases its ArraySize by 50%. find detailed explanation ArrayList vs Vector .

Q17 What is the difference between HashMap and Hashtable ?

It is one of the most popular collections interview question for java developer . Make sure you go through this once before appearing for the interview .

Main differences between HashMap and Hashtable are :

a. HashMap allows one null key and any number of null values while Hashtable does not allow null keys and null values.

b. HashMap is not synchronized or thread-safe while Hashtable is synchronized or thread-safe .

find detailed explanation here Hashtable vs HashMap in Java

Q18 What is the difference between peek(),poll() and remove() method of the Queue interface ?

Both poll() and remove() method is used to remove head object of the Queue. The main difference lies when the Queue is empty().

If Queue is empty then poll() method will return null . While in similar case , remove() method will throw NoSuchElementException .

peek() method retrieves but does not remove the head of the Queue. If queue is empty then peek() method also returns null.

Q19 What is the difference between Iterator and ListIterator.

Using Iterator we can traverse the list of objects in forward direction . But ListIterator can traverse the collection in both directions that is forward as well as backward.

Q20 What is the difference between Array and ArrayList in Java ?

This question checks whether student understand the concept of static and dynamic array. Some main differences between Array and ArrayList are :

a. Array is static in size while ArrayList is dynamic in size.

b. Array can contain primitive data types while ArrayList can not contain primitive data type

Q21 What is the difference between HashSet and TreeSet ?

Main differences between HashSet and TreeSet are :

a. HashSet maintains the inserted elements in random order while TreeSet maintains elements in the sorted order

b. HashSet can store null object while TreeSet can not store null object.

find detailed explanation here TreeSet vs HashSet in Java

Q22 Write java code showing insertion,deletion and retrieval of HashMap object ?

Do it yourself (DIY) , if found any difficulty or doubts then please mention in the comments.

Q23 What is the difference between HashMap and ConcurrentHashMap ?

This is also one of the most popular java collections interview question . Make sure this question is in your to do list before appearing for the interview .

Main differences between HashMap and ConcurrentHashMap are :

a. HashMap is not synchronized while ConcurrentHashMap is synchronized.

b. HashMap can have one null key and any number of null values while ConcurrentHashMap does not allow null keys and null values .

Q24 Arrange the following in the ascending order (performance):

HashMap , Hashtable , ConcurrentHashMap and Collections.SynchronizedMap

Hashtable < Collections.SynchronizedMap < ConcurrentHashMap < HashMap

Q25 How HashMap works in Java ?

This is one of the most important question for java developers. HashMap works on the principle of Hashing .

Q26 What is the difference between LinkedList and ArrayList in Java ?

Main differences between LinkedList and ArrayList are :

a. LinkedList is the doubly linked list implementation of list interface , while , ArrayList is the resizable array implementation of list interface.

b. LinkedList can be traversed in the reverse direction using descendingIterator() method provided by the Java Api developers , while , we need to implement our own method to traverse ArrayList in the reverse direction . find the detailed explanation here

27 What are Comparable and Comparator interfaces ? List the difference between them ?

We already explained what is comparable and comparator interface in detail along with examples here, Comparable vs Comparator in Java

Q28 Why Map interface does not extend the Collection interface in Java Collections Framework ?

One liner answer : Map interface is not compatible with the Collection interface.

Explanation : Since Map requires key as well as value , for example , if we want to add key-value pair then we will use put(Object key , Object value) . So there are two parameters required to add element to the HashMap object . In Collection interface add(Object o) has only one parameter.

The other reasons are Map supports valueSet , keySet as well as other appropriate methods which have just different views from the Collection interface.

Q29 When to use ArrayList and when to use LinkedList in application?

ArrayList has constant time search operation O(1) .Hence, ArrayList is preferred when there are more get() or search operation .

Insertion , Deletion operations take constant time O(1) for LinkedList. Hence, LinkedList is preferred when there are more insertions or deletions involved in the application.

Q30 Write the code for iterating the list in different ways in java ?

There are two ways to iterate over the list in java :

a. using Iterator

b. using for-each loop

Coding part : Do it yourself (DIY) , in case of any doubts or difficulty please mention in the comments .

Advance Level (3+ yrs): Java Collections Interview Questions and Answers

Q31 How HashSet works internally in java ?

This is one of the popular interview question . HashSet internally uses HashMap to maintain the uniqueness of elements.

Q33 How HashMap works in Java ?

We are repeating this question , as it is one of the most important question for java developer.HashMap works on the principle of Hashing

35 What is BlockingQueue in Java Collections Framework?

BlockingQueue implements the java.util.Queue interface . BlockingQueue supports operations that wait for the queue to become non-empty when retrieving an element , and wait for space to become available in the queue when storing an element .

It does not accept null elements.

Blocking queues are primarily designed for the producer-consumer problems.

BlockingQueue implementations are thread-safe and can also be used in inter-thread communications.

This concurrent Collection class was added in jdk 1.5

Q36 How TreeMap works in Java ?

TreeMap internally uses Red-Black tree to sort the elements in natural order.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INTERFACE IN JAVA

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Interface in Java

Interface

Example of Interface

Multiple inheritance by Interface

Why multiple inheritance is supported in Interface while it is not supported in case of class.

Marker Interface

Nested Interface

An interface in java is a blueprint of a class. It has static constants and abstract methods.

The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also represents IS-A relationship.

It cannot be instantiated just like abstract class.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

It is used to achieve abstraction.

By interface, we can support the functionality of multiple inheritance.

It can be used to achieve loose coupling.

Java 8 Interface Improvement

Since Java 8, interface can have default and static methods which is discussed later.

Internal addition by compiler

The java compiler adds public and abstract keywords before the interface method. More, it adds public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and methods are public and abstract.

java Interface Example

In this example, Printable interface has only one method, its implementation is provided in the A class.

interface printable{

void print();

}

class A6 implements printable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

A6 obj = new A6();

obj.print();

}

}

Java Interface Example: Drawable

In this example, Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In real scenario, interface is defined by someone but implementation is provided by different implementation providers. And, it is used by someone else. The implementation part is hidden by the user which uses the interface.

File: TestInterface1.java

//Interface declaration: by first user

interface Drawable{

void draw();

}

//Implementation: by second user

class Rectangle implements Drawable{

public void draw(){System.out.println("drawing rectangle");}

}

class Circle implements Drawable{

public void draw(){System.out.println("drawing circle");}

}

//Using interface: by third user

class TestInterface1{

public static void main(String args[]){

Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()

d.draw();

}}

Java Interface Example: Bank

Let's see another example of java interface which provides the implementation of Bank interface.

File: TestInterface2.java

interface Bank{

float rateOfInterest();

}

class SBI implements Bank{

public float rateOfInterest(){return 9.15f;}

}

class PNB implements Bank{

public float rateOfInterest(){return 9.7f;}

}

class TestInterface2{

public static void main(String[] args){

Bank b=new SBI();

System.out.println("ROI: "+b.rateOfInterest());

}}

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.

interface Printable{

void print();

}

interface Showable{

void show();

}

class A7 implements Printable,Showable{

public void print(){System.out.println("Hello");}

public void show(){System.out.println("Welcome");}

public static void main(String args[]){

A7 obj = new A7();

obj.print();

obj.show();

}

}

Multiple inheritance is not supported through class in java but it is possible by interface, why?

As we have explained in the inheritance chapter, multiple inheritance is not supported in case of class because of ambiguity. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class. For example:

interface Printable{

void print();

}

interface Showable{

void print();

}

class TestInterface3 implements Printable, Showable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

TestInterface3 obj = new TestInterface3();

obj.print();

}

}

Interface inheritance

A class implements interface but one interface extends another interface .

interface Printable{

void print();

}

interface Showable extends Printable{

void show();

}

class TestInterface4 implements Showable{

public void print(){System.out.println("Hello");}

public void show(){System.out.println("Welcome");}

public static void main(String args[]){

TestInterface4 obj = new TestInterface4();

obj.print();

obj.show();

}

}

Java 8 Default Method in Interface

Since Java 8, we can have method body in interface. But we need to make it default method. Let's see an example:

File: TestInterfaceDefault.java

interface Drawable{

void draw();

default void msg(){System.out.println("default method");}

}

class Rectangle implements Drawable{

public void draw(){System.out.println("drawing rectangle");}

}

class TestInterfaceDefault{

public static void main(String args[]){

Drawable d=new Rectangle();

d.draw();

d.msg();

}}

What is marker or tagged interface?

An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

//How Serializable interface is written?

public interface Serializable{

}

Nested Interface in Java

Note: An interface can have another interface i.e. known as nested interface. We will learn it in detail in the nested classes chapter. For example:

interface printable{

void print();

interface MessagePrintable{

void msg();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXCEPTION HANDLING

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors.It is mainly used to handle checked exceptions.

What is difference between Checked Exception and Unchecked Exception?

1)Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException,SQLException etc. Checked exceptions are checked at compile-time.

2)Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException,NullPointerException etc. Unchecked exceptions are not checked at compile-time.

What is the base class for Error and Exception?

Throwable.

Is it necessary that each try block must be followed by a catch block?

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. And whatever exceptions are likely to be thrown should be declared in the throws clause of the method.

What is finally block?

finally block is a block that is always executed.more details...

Can finally block be used without catch?

Yes, by try block. finally must be followed by either try or catch.more details...

Is there any case when finally will not be executed?

finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).more details...

What is difference between throw and throws?

throw keyword throws keyword

1)throw is used to explicitly throw an exception. throws is used to declare an exception.

2)checked exceptions can not be propagated with throw only. checked exception can be propagated with throws.

3)throw is followed by an instance. throws is followed by class.

4)throw is used within the method. throws is used with the method signature.

5)You cannot throw multiple exception You can declare multiple exception e.g. public void method()throws IOException,SQLException.

more details...

Can an exception be rethrown?

Yes.

Can subclass overriding method declare an exception if parent class method doesn't throw an exception ?

Yes but only unchecked exception not checked.

more details...

What is exception propagation ?

Forwarding the exception object to the invoking method is known as exception propagation.

What is the meaning of immutable in terms of String?

The simple meaning of immutable is unmodifiable or unchangeable. Once string object has been created, its value can't be changed.

Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JAVA IMP PROGRAM QUESTION

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Verify a number is Even/Odd

import java.util.Scanner;

public class EvenOdd{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter a number which you want to check whether that is even or odd");

int n = in.nextInt();

if(n%2==0){

System.out.println(n+" is an even number.");

}else{

System.out.println(n+" is an odd number.");

}

}

}

import java.util.Scanner;

public class EvenOdd{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter a number which you want to check whether that is even or odd");

int n = in.nextInt();

if(n%2==0){

System.out.println(n+" is an even number.");

}else{

System.out.println(n+" is an odd number.");

}

}

}

Output

Enter a number which you want to check whether that is even or odd

4

4 is an even number.

1

2

3

Enter a number which you want to check whether that is even or odd

4

4 is an even number.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Swapping Numbers without using 3rd variable

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Scanner;

public class Swapping{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter the 1st number: ");

int x = in.nextInt();

System.out.println("Enter the 2nd number: ");

int y = in.nextInt();

System.out.println("Initial value of x: "+x+" and y: "+y);

x = x+y;

y = x-y;

x = x-y;

System.out.println("After swapping value of x: "+x+" and y: "+y);

}

}

import java.util.Scanner;

public class Swapping{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter the 1st number: ");

int x = in.nextInt();

System.out.println("Enter the 2nd number: ");

int y = in.nextInt();

System.out.println("Initial value of x: "+x+" and y: "+y);

x = x+y;

y = x-y;

x = x-y;

System.out.println("After swapping value of x: "+x+" and y: "+y);

}

}

Output

Enter the 1st number:

43

Enter the 2nd number:

56

Initial value of x: 43 and y: 56

After swapping value of x: 56 and y: 43

Enter the 1st number:

43

Enter the 2nd number:

56

Initial value of x: 43 and y: 56

After swapping value of x: 56 and y: 43

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

How to get the prime numbers between a given range.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package javaTutorial;

import java.util.ArrayList;

import java.util.Scanner;

public class GetPrimeNumbers{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter a number from which you want prime number: ");

int p1 = in.nextInt();

System.out.println("Enter one more number till which you want prime number: ");

int p2 = in.nextInt();

ArrayList&lt;Integer&gt; prime = new ArrayList&lt;Integer&gt;();

int i=2;

for(int p=p1; p&lt;=p2; p++){

i=2;

for(; i&lt;10; i++){

if(p%i==0 &amp;&amp; p!=i){

break;

}

}

if(i==10){

prime.add(p);

}

}

System.out.println("Prime numbers between "+p1+" and "+p2+" are: ");

for(int j=0; j&lt;prime.size(); j++){

System.out.print(prime.get(j).toString()+", ");

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PRIME NUMBER

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package javaTutorial;

import java.util.ArrayList;

import java.util.Scanner;

public class GetPrimeNumbers{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter a number from which you want prime number: ");

int p1 = in.nextInt();

System.out.println("Enter one more number till which you want prime number: ");

int p2 = in.nextInt();

ArrayList&lt;Integer&gt; prime = new ArrayList&lt;Integer&gt;();

int i=2;

for(int p=p1; p&lt;=p2; p++){

i=2;

for(; i&lt;10; i++){

if(p%i==0 &amp;&amp; p!=i){

break;

}

}

if(i==10){

prime.add(p);

}

}

System.out.println("Prime numbers between "+p1+" and "+p2+" are: ");

for(int j=0; j&lt;prime.size(); j++){

System.out.print(prime.get(j).toString()+", ");

}

}

}

Output

Enter a number from which you want prime number:

10

Enter one more number till which you want prime number:

30

Prime numbers between 10 and 30 are:

11, 13, 17, 19, 23, 29,

Enter a number from which you want prime number:

10

Enter one more number till which you want prime number:

30

Prime numbers between 10 and 30 are:

11, 13, 17, 19, 23, 29,

Check a number is prime or not.

Note- A number is prime if it is not divisible by any other number except itself.

import java.util.Scanner;

public class PrimeNumber{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter a number greater than 2 which you want to check whether that number is prime or not: ");

int p = in.nextInt();

int i=2;

for(; i&lt;10; i++){

if(p%i==0 &amp;&amp; p!=i){

System.out.println("Entered number "+p+" is not a prime number.");

break;

}

}

if(i==10){

System.out.println("Entered number "+p+" is a prime number.");

}

}

}

import java.util.Scanner;

public class PrimeNumber{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter a number greater than 2 which you want to check whether that number is prime or not: ");

int p = in.nextInt();

int i=2;

for(; i&lt;10; i++){

if(p%i==0 &amp;&amp; p!=i){

System.out.println("Entered number "+p+" is not a prime number.");

break;

}

}

if(i==10){

System.out.println("Entered number "+p+" is a prime number.");

}

}

}

Output

Enter a number greater than 2 which you want to check whether that number is prime or not:

139

Entered number 139 is a prime number.

1

2

3

Enter a number greater than 2 which you want to check whether that number is prime or not:

139

Entered number 139 is a prime number.

Check if a number is Armstrong or not.

Note- A number is armstrong if the sum of the cubes of digit of number is equal to the number.

ex- 407 = 4\*4\*4 + 0\*0\*0 + 7\*7\*7

import java.util.Scanner;

public class ArmstrongNum{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter a number which you want to check whether that is armstrong or not: ");

int n = in.nextInt();

int a = n, r=0, s=0;

while(a!=0){

r = a%10;

a = a/10;

s = s + r\*r\*r;

}

if(s==n){

System.out.println("Number "+n+" is an armstrong number.");

}else{

System.out.println("Number "+n+" is not an armstrong number.");

}

}

}

import java.util.Scanner;

public class ArmstrongNum{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter a number which you want to check whether that is armstrong or not: ");

int n = in.nextInt();

int a = n, r=0, s=0;

while(a!=0){

r = a%10;

a = a/10;

s = s + r\*r\*r;

}

if(s==n){

System.out.println("Number "+n+" is an armstrong number.");

}else{

System.out.println("Number "+n+" is not an armstrong number.");

}

}

}

Output

Enter a number which you want to check whether that is armstrong or not:

407

Number 407 is an armstrong number.

1

2

3

Enter a number which you want to check whether that is armstrong or not:

407

Number 407 is an armstrong number.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Floyd Triangle

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Note- Floyd Triangle is like

1

2 3

4 5 6

7 8 9 10

————

Code-

import java.util.Scanner;

public class FloydTriangle{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter the number of rows which you want in your Floyd Triangle: ");

int r = in.nextInt();

int n=0;

for(int i=0; i&lt;r; i++){

for(int j=0; j&lt;=i; j++){

System.out.print(++n+" ");

}

System.out.println();

}

}

}

import java.util.Scanner;

public class FloydTriangle{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter the number of rows which you want in your Floyd Triangle: ");

int r = in.nextInt();

int n=0;

for(int i=0; i&lt;r; i++){

for(int j=0; j&lt;=i; j++){

System.out.print(++n+" ");

}

System.out.println();

}

}

}

Output

Enter the number of rows which you want in your Floyd Triangle:

5

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

1

2

3

4

5

6

7

Enter the number of rows which you want in your Floyd Triangle:

5

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Palindrome of String or reverse a String.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Scanner;

public class PalindromeString{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter the string which you want to check whether that is palindrome or not: ");

String s = in.next();

String r = "";

for(int i=s.length()-1; i&gt;=0; i--){

r = r+s.charAt(i);

}

System.out.println("Reverse of entered string "+s+" is "+r);

if(r.equals(s)){

System.out.println("String "+s+" is palindrome.");

}else{

System.out.println("String "+s+" is not palindrome.");

}

}

}

Output

Enter the string which you want to check whether that is palindrome or not:

selenium

Reverse of entered string selenium is muineles

String selenium is not palindrome.

Enter the string which you want to check whether that is palindrome or not:

selenium

Reverse of entered string selenium is muineles

String selenium is not palindrome.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Binary Search

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Java

import java.util.Arrays;

import java.util.Scanner;

public class BinarySearch{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter the size of the array which should be greater than zero else it will throw InputMismatchException : ");

int size = in.nextInt();

int[] array = new int[size];

System.out.println("Enter the elements of the array: ");

for(int i=0; i&lt;size; i++){

array[i] = in.nextInt();

}

System.out.println("Enter the search element: ");

int s = in.nextInt();

Arrays.sort(array); //binary search will work on sorted array only so sort first

int first, last, middle;

first=0;

last = size-1;

middle = (first+last)/2;

int i=0;

for(; i&lt;size; i++){

if(s&gt;array[middle]){

first = middle+1;

}else if(s&lt;array[middle]){

last = middle-1;

}else{

printArray(array);

System.out.println("Element "+s+" found in the array.");

break;

}

middle= (first+last)/2;

}

if(i==size){

printArray(array);

System.out.println("Element "+s+" is not found in the array");

}

}

public static void printArray(int[] a){

System.out.println("Array of elements: ");

System.out.print("{");

for(int i:a){

System.out.print(i+",");

}

System.out.println("}");

}

}

import java.util.Arrays;

import java.util.Scanner;

public class BinarySearch{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

System.out.println("Enter the size of the array which should be greater than zero else it will throw InputMismatchException : ");

int size = in.nextInt();

int[] array = new int[size];

System.out.println("Enter the elements of the array: ");

for(int i=0; i&lt;size; i++){

array[i] = in.nextInt();

}

System.out.println("Enter the search element: ");

int s = in.nextInt();

Arrays.sort(array); //binary search will work on sorted array only so sort first

int first, last, middle;

first=0;

last = size-1;

middle = (first+last)/2;

int i=0;

for(; i&lt;size; i++){

if(s&gt;array[middle]){

first = middle+1;

}else if(s&lt;array[middle]){

last = middle-1;

}else{

printArray(array);

System.out.println("Element "+s+" found in the array.");

break;

}

middle= (first+last)/2;

}

if(i==size){

printArray(array);

System.out.println("Element "+s+" is not found in the array");

}

}

public static void printArray(int[] a){

System.out.println("Array of elements: ");

System.out.print("{");

for(int i:a){

System.out.print(i+",");

}

System.out.println("}");

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Bubble Sort

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Java

package SeleniumMakeItEasy;

import org.apache.commons.lang3.ArrayUtils;

public class BubbleSort{

public static void main(String[] args){

int[] a = {2,3,2,5,3,3,6,1,2,5};

int l = a.length;

for(int i=0;i&lt;l; i++){

for(int j=0; j&lt;l-1; j++){

if(a[j]&gt;a[j+1]){

a[j] = a[j] + a[j+1];

a[j+1] = a[j] - a[j+1];

a[j] = a[j] - a[j+1];

}else if(a[j]==a[j+1]){

a = ArrayUtils.remove(a,j);

l = a.length;

}

}

}

for(int s: a){

System.out.println(s);

}

}

}

package SeleniumMakeItEasy;

import org.apache.commons.lang3.ArrayUtils;

public class BubbleSort{

public static void main(String[] args){

int[] a = {2,3,2,5,3,3,6,1,2,5};

int l = a.length;

for(int i=0;i&lt;l; i++){

for(int j=0; j&lt;l-1; j++){

if(a[j]&gt;a[j+1]){

a[j] = a[j] + a[j+1];

a[j+1] = a[j] - a[j+1];

a[j] = a[j] - a[j+1];

}else if(a[j]==a[j+1]){

a = ArrayUtils.remove(a,j);

l = a.length;

}

}

}

for(int s: a){

System.out.println(s);

}

}

}