Contents

[**Java Overview** 2](#_Toc507923340)

[**Programming Basics** 2](#_Toc507923341)

[**Java Introduction** 5](#_Toc507923342)

[**Javac(compiler), JVM, Java(interpreter) and JIT(compiler)** 6](#_Toc507923343)

[**Java Features** 11](#_Toc507923344)

[**A Java Program** 14](#_Toc507923345)

[**Simple Class and Object** 19](#_Toc507923346)

[**Data Types** 21](#_Toc507923347)

[**Literals and Escape Sequence** 23](#_Toc507923348)

[**Variables - (Static,Instance,Local)** 24](#_Toc507923349)

[**Access Modifiers** 28](#_Toc507923350)

[**Non-Access Modifiers - (static, final , abstract….)** 31](#_Toc507923351)

[**Operators** 36](#_Toc507923352)

[**Loops** 38](#_Toc507923353)

[**Conditional Structure** 41](#_Toc507923354)

[**Array** 43](#_Toc507923355)

[**Date/Time – old** 56](#_Toc507923356)

[**Regular Expression** 64](#_Toc507923357)

[**Methods and Constructors** 72](#_Toc507923358)

[**Nested Class** 78](#_Toc507923359)

[**Inheritance - (Single , Multilevel, Hierarchical)** 82](#_Toc507923360)

[**Polymorphism** 86](#_Toc507923361)

[**Abstraction** 88](#_Toc507923362)

[**Encapsulation** 90](#_Toc507923363)

[**Interface** 91](#_Toc507923364)

[**Package** 96](#_Toc507923365)

[**Collections** 98](#_Toc507923366)

|  |
| --- |
|  |
| **Simple Class and Object** |
|  |

|  |
| --- |
|  |
| What is Class? |
| A class is a blue print from which individual objects are created. A class can contain fields and methods to describe the behavior of an object. |
| What is Object? |
| * Object is a runtime entity and consist states and behaviours. * It’s state is stored in fields and behavior is shown via methods. * An object is an instance of a class.   Example:  A dog has states - color, name, breed as well as  behaviors – wagging the tail, barking, eating. |
| List the different ways for creating an Object for a class? |
| 1. Using the new keyword   It is the most common and regular way to create an object and actually very simple one also. By using this method we can call whichever constructor we want to call (no-arg constructor as well as parametrised).  Employee emp1 = new Employee();   1. Using Class.newInstance() method   We can also use the newInstance() method of the Class class to create objects, This newInstance() method calls the no-arg constructor to create the object.  We can create objects by newInstance() in any of the following way.   1. Employee emp2 = (Employee) Class.forName("org.programming.mitra.exercises.Employee")   .newInstance();   1. Employee emp2 = Employee.class.newInstance(); 2. Using newInstance() method of Constructor class   Similar to the newInstance() method of Class class, There is one newInstance() method in the java.lang.reflect.  Constructor class which we can use to create objects.  We can also call parameterized constructor, and private constructor by using this newInstance() method.  Constructor<Employee> constructor = Employee.class.getConstructor();  Employee emp3 = constructor.newInstance();   1. Using clone() method   Whenever we call clone() on any object JVM actually creates a new object for us and copy all content of the previous object into it.  Creating an object using clone method does not invoke any constructor.  To use clone() method on an object we need to implements Cloneable and define clone() method in it.  Employee emp4 = (Employee) emp3.clone();   1. Using deserialization   Object deserialization is nothing but creating an object from its serialized form.  ObjectInputStream inStream = new ObjectInputStream(anInputStream );  MyObject object = (MyObject) inStream.readObject(); |
| List the three steps for creating an Object for a class? |
| import java.util.Date;  class DateApp {  public static void main (String args[]) {  Date today = new Date();  System.out.println(today);  }  }  An Object is first declared, then instantiated and then it is initialized.   * Declaration: A variable declaration with a variable name with an object type.   Date today;  Or  Date today ~~= new Date();~~   * Instantiation: The new operator instantiates a new object by allocating memory for it.   new requires a single argument: a constructor method for the object to be created. The constructor method is responsible for initializing the new object.  ~~Date today =~~ new ~~Date();~~   * Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.   ~~Date today = new~~ Date(); |
| What is Singleton class? |
| Singleton class control object creation, limiting the number to one but allowing the flexibility to create more objects if the situation changes. |
| **Data Types** |
|  |
| What is Data type ? |
| Each variable has type associated with it called as it is datatype. Data type decides the type and range of values that can be stored within variable (named memory location) and the set of operations that can be applied to the variable.  There are two data types available in Java:   * Primitive Datatypes * Reference/Object Datatypes |
| What is Primitive Datatypes? |
| Primitive (very basic) datatypes are predefined datatypes by the language.  Primitive variables store the actual values  There are 8 primitive types:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Datatype | Minimum value | Maximum value | Default value | Example | | Byte - 8-bit | -128 (-2^7) | 127 (2^7 -1) | 0 | byte a = 100;  byte b = -50; | | Short - 16-bit | -32,768 (-2^15) | 32,767 (2^15 -1) | 0 | short s = 10000; | | Int- 32-bit | (-2^31) | (2^31 -1) | 0 | int a = 100000;  int b = -200000 | | Long- 64-bit | (-2^63) | (-2^63 -1) | 0L | long a = 100000L,  long b = -200000L; | | Char- 16-bit | '\u0000' (or 0) | '\uffff' (or 65,535 inclusive) | Space | char letterA ='A' | | Float  - single-precision 32-bit |  |  | 0.0 | float f1 = 234.5f | | Double- double-precision 64-bit |  |  | 0.0 | double d1 = 123.4 | | Boolean-1 bit | true and false | true and false | False | boolean one = true | |
| What is Reference/Object Datatypes? |
| Reference variables store the addresses of the objects they refer to. Actual object stores in Heap memory.  Class, array comes under reference datatype.  Default value of any reference variable is null. |
| **Literals and Escape Sequence** |
|  |
| What is Literal ? |
| * A Literal is the source code representation of a fixed value. * literals are represented directly in your code without requiring computation. * Example : ~~int decimal =~~ 100; * byte, int, long, and short can be expressed in decimal(base 10), hexadecimal(base 16) or octal(base 8) number systems as well.   int decimal = 100;  int octal = 0144;  int hexa = 0x64;   * String literals Example:   ~~String s =~~ "Hello World"  ~~String s2 =~~ "two\nlines"   * String literals Example:   ~~char a =~~ 'A';   * String and char types of literals can contain any Unicode characters. For example:   char a = '\u0001';  String a = "\u0001";   * Java language supports few special escape sequences (\n, \r) for String and char literals as well.   Example : ~~String s2 =~~ "two\nlines" |
| What are Escape characters?  What are Escape sequences?  What are Escape codes? |
| * Escape characters (also called escape sequences or escape codes) in general are used to signal an alternative interpretation of a series of characters. * They are preceded by a backslash (\) * Java language supports few special escape sequences for String and char literals as well. |
| **Variables - (Static,Instance,Local)** |
|  |
| What is Variable ? |
| * A variable is a named memory location to store values that can be changed during program execution. * Each variable has type associated with it, called as datatype. Data type decides the type and range of values that can be stored within variable (named memory location) and the set of operations that can be applied to the variable. |
| How many types of Variable we have in Java? |
| There are three kinds of variables in Java:   * Local variables * Instance variables * Class/Static variables |
| What is a Local Variable? |
| Variables defined inside methods, constructors or blocks are called local variables.   * Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block. * Access modifiers cannot be used for local variables. * Local variables are visible only within the declared method, constructor or block. * Local variables are implemented at stack level internally. * There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use. |
| What is Instance Variable? |
| Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded.   * Instance variables are declared in a class, but outside a method, constructor or any block. * When a space is allocated for an object in the heap, a slot for each instance variable value is created. * Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. * Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class. * Instance variables can be declared in class level before or after use. * Access modifiers can be given for instance variables. * The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers. * Instance variables have default values.   For numbers, the default value is 0,  For Booleans it is false  For object references it is null.   * Values can be assigned during the declaration or within the constructor. * Instance variables can be accessed directly by calling the variable name inside the class * However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name.   ObjectReference.VariableName. |
| What is a Class Variable?  What is a Static Variable? |
| These are variables declared with in a class, outside any method, with the static keyword.   * Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block. * There would only be one copy of each class variable per class, regardless of how many objects are created from it. * ~~Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.~~ * Static variables are stored in the static memory. * Static variables are created when the program starts and destroyed when the program stops. * Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class. * Default values are same as instance variables.   For numbers, the default value is 0  For Booleans, it is false  For object references, it is null.  Values can be assigned during the declaration or within the constructor.  Additionally, values can be assigned in special static initializer blocks.   * If the static variables are not public and final, the naming syntax is the same as instance and local variables. * Static variables can be accessed by calling with the class name ClassName.VariableName. * ~~When declaring class variables as public static final, then variable names (constants) are all in upper case.~~ |
| How variables stores? |
| * Static variables are stored in the static memory / Method Area.   Contains default values too.   * Object gets memory in Heap and object reference in Stack.   Object reference contains default value(null).   * Since Object gets memory in Heap , so does it’s object variable (instance variable). Contains default values too. * Local variables stores in Stack.   Contains No default values. |
| **Access Modifiers** |
|  |
| What do you mean by Modifier? |
| Modifiers are keywords that you add to class, method, constructor, variables to change their meanings or way of working.  Java language has a wide variety of modifiers, including the following:   * Access Modifiers * Non Access Modifiers |
| What do you mean by Access Modifier? |
| Modifiers are keywords that you add to class, method, constructor, variables to set access levels.  The four access levels are:   * Private * Default(No modifier mentioned) * Protected * Public  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Visibility** | **Private**  **(within Class)** | **Default**  **(within Package)** | **Protected**  **(within Same Package and Sub classes of different package)** | **Public**  **(Everywhere within the Java project)** | | **Same Class** | **Accessible** | **Accessible** | **Accessible** | **Accessible** | | **Same Package** |  | **Accessible** | **Accessible** | **Accessible** | | **Different Package - Sub Class** |  |  | **Accessible** | **Accessible** | | **Different Package - Non Sub Class** |  |  |  | **Accessible** |  * Interface:  1. Can only be public or default. 2. Variables declared in interface can be public, static and final. Even if we don’t write in the code, it will be considered public, static and final by default. 3. All the interface methods can be and by default abstract and public. Even if we don’t write in the code.      * Inheritance:  1. Methods declared public in a superclass also must be public in all subclasses. 2. Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private. 3. Methods declared private are not inherited at all, so there is no rule for them. |
| Can I have a private class?  Can I have a protected class? |
| * Class:  1. Can only be public or default, unless it’s inner class. |
| Can I have a private interface?  Can I have a protected interface? |
| * Interface:  1. Can only be public or default. |
| What is the access level of interface fields if I do not append any modifier for them?  Can I have interface fields as private?  Can I have interface fields as protected? |
| * Variables declared in interface can be public, static and final. Even if we don’t write in the code, it will be considered public, static and final by default. |
| What is the access level of interface method if I do not append any modifier for them?  Can I have interface method as private?  Can I have interface method as protected? |
| * All the interface methods can be and by default abstract and public. Even if we don’t write in the code. |
| I have a Methods declared protected in a superclass can I declare it as private in subclasses? |
| * No * Inheritance:  1. Methods declared public in a superclass also must be public in all subclasses. 2. Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private. 3. Methods declared private are not inherited at all, so there is no rule for them. |
| How can I access private variables? |
| Variables that are declared private can be accessed outside the class using multiple ways, most popular are:   * 1. Using Non-private setter and getter methods are present in the class.   2. Using Constructor to set the value. |
| **Non-Access Modifiers - (static, final , abstract….)** |
|  |
| What do you mean by Modifier? |
| Modifiers are keywords that you add to class, method, constructor, variables to change their meanings or way of working.  Java language has a wide variety of modifiers, including the following:   * Access Modifiers * Non Access Modifiers |
| What do you mean by Non-Access Modifier? |
| Non-Access Modifiers are keywords that you add to class, method, and constructor, variables to change their meanings or way of working but does not change access level directly.  Below are some popular Non-access modifiers:   1. static 2. final 3. abstract 4. synchronized 5. volatile |
| What is the use of static keyword? |
| static keyword use to create methods and variables that directly belongs to Class. |
| Can I have static class? |
| Yes , but the class should be nested class |
| What is a Class Variable?  What is a Static Variable? |
| These are variables declared with in a class, outside any method, constructor with the static keyword. There would only be one copy of each class variable per class, regardless of how many objects are created from it.   * Static variables are stored in the static memory. * Static variables are created when the program starts and destroyed when the program stops. * Default values are same as instance variables.   For numbers, the default value is 0  For Booleans, it is false  For object references, it is null.  Values can be assigned during the declaration or within the constructor.  Additionally, values can be assigned in special static initializer blocks.   * Static variables can be accessed by calling with the class name ClassName.VariableName outside the class. |
| What is a Class Method?  What is a Static Method? |
| If we have static keyword associated with any method declaration, it is known as static method. A static method belongs to the class rather than object of a class.   * Static method can be invoked by calling with the class name outside the class. No need to create instance/object of the class.   ClassName. StaticMethodName();   * Static method can be invoked outside the class by object as well.   ClassName obj = new ClassName();  obj.StaticMethodName();   * Static method can access static data member and can change the value of it. * Important: Static methods cannot use any instance variables of the class they are defined in. * Static methods can call instance method indirectly only.   new ClassName().instanceMethodName;   * One rule-of-thumb: ask yourself "does it make sense to call this method, even if no Obj has been constructed yet?" If so, it should definitely be static. Example: for utility classes. |
| What is the use of Final Modifier? |
| The final modifier restricts future changes on classes, methods and variables.  A constructor cannot be final. |
| What is Final Variable does? |
| A final variable can be explicitly initialized only once.   * A reference variable declared final can never be reassigned to refer to a different object. However, the data within the object can be changed. So, the state of the object can be changed but not the reference. * final modifier often is used with static to make the constant class variable. |
| What is Final Method does? |
| A final method cannot be overridden by any subclasses.  A constructor cannot be final. |
| What is Final class does? |
| The main purpose of using a class being declared as final is to prevent the class from being subclassed. |
| What is the use of abstract keyword? |
| The abstract modifier for creating abstract (existing in idea) classes and methods. |
| What Abstract class does? |
| If a class is declared as abstract then the sole purpose is for the class to be extended.   * An abstract class may contain zero or more abstract methods as well normal methods. * An abstract class does not need to contain abstract methods. * Abstract class can contain normal methods as well as undefined method (with abstract methods), this make it different from interface. * An abstract class can never be instantiated. * A class cannot be both abstract and final, both are opposite of each other. |
| What Abstract method does? |
| An abstract method declares without any implementation. The methods body (implementation) is provided by the subclass.   * Abstract methods can never be final or strict. * Any class that extends an abstract class must implement all the abstract methods of the super class, unless the subclass is also an abstract class. |
| What is Synchronized Modifier? |
| The synchronized keyword used to indicate that a method can be accessed by only one thread at a time.  With synchronized method :    With synchronized method : |
| What Transient keyword does? |
| Serialization in java is a mechanism of writing the state of an object into a byte stream.  If you define any data member as transient, it will not be serialized.   * transient and static : Since static fields are not part of state of the object, there is no use/impact of using transient keyword with static variables. However there is no compilation error. * transient and final : final variables are directly serialized by their values, so there is no use/impact of declaring final variable as transient. There is no compile-time error though. |
| What volatile modifier does ? |
| volatile modifier guarantees that any thread that reads a field will see the most recently written value.  the value of a volatile field becomes visible to all readers (other threads in particular) after a write operation completes on it. Without volatile, readers could see some non-updated value. |
| **Operators** |
|  |
| What is Conditional Operator (? :) in Java ?  What is Ternary Operator (? :) in Java ? |
| Conditional operator, can be used as an alternative of if/then/else syntax.  Operator let's you assign a value to a variable based on a Boolean output of the condition and true/false expression. That’s the reason it can be used to identify value that should be assigned to the variable  Syntax :   1. return\_value = (true-false condition) ? (if true expression) : (if false expression); 2. return\_value = (true-false condition)   ? (if true expression)  : (if false expression); |
| What instanceof Operator does ? |
| This operator is used only for object reference variables, to checks whether the object is of a particular type (class type or interface type).  instanceof operator is written as:  ( Object reference variable ) instanceof (class/interface type) |
| List down some of the operators ? |
| Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first. |
| **Loops** |
|  |
| What are loop statements? |
| A loop statement allows us to execute a statement or group of statements multiple times  Java programming language provides the following types of loop:   1. while 2. do...while 3. for 4. for each |
| What while loop does? |
| Repeats target statements while a given condition is true. It tests the condition before executing the loop body.  When the condition becomes false, program control passes to the line immediately following the loop. |
| What do…while loop does? |
| Repeats target statements while a given condition is true. It tests the condition at the end of executing the loop body.  When the condition becomes false, program control passes to the line immediately following the loop. |
| What for loop does? |
| Very first time it set iteration counter and execute target statements multiple times if Boolean expression is true.  Second time onwards it executes update counter statement and executes target statements multiple times if Boolean expression is true.   * A for loop is useful when you know how many times a task is to be repeated. |
| What Enhanced for loop does?  What for each loop does? |
| Traverse collection of elements including arrays.  Syntax  Following is the syntax of enhanced for loop: |
| What are Loop Control Statements? |
| Loop control statements change execution from its normal sequence.  Java supports the following control statements.   1. break 2. continue |
| What break statement does? |
| Skip remaining statements (of the loop/switch) and terminates (jumps out of) the loop (while,doWhile,for,forEach) or switch statement and transfers execution to the statement immediately following the loop or switch. |
| What continue statement does? |
| Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| **Conditional Structure** |
|  |
| What are Decision making structures?  What are conditional structures?  What are conditional statements? |
| Decision making structures have one or more conditions to be evaluated, along with statements to be executed if the condition is true and optionally, other statements to be executed if the condition is false.   |  |  | | --- | --- | | **Statement** | **Description** | | **if statement** | An **if statement** consists of a boolean expression followed by one or more statements. | | **if...else statement** | An **if statement** can be followed by an **else statement**, which executes when the boolean expression is false. | | **Ladder if...else statements** | An **if...else** statement can be followed by multiple **if...else** statement. | | **Nested if statements** | You can use **if** or **if..else** statement inside **if** or **else** block. | | **switch statement** | A switch statement allows a variable to be tested for equality matching against a list of values called a case.  Once matched case found, statements associated to the case and follow cases and defaults (optional) executed until a break statements comes.  If no match case found, control moves to the default (optional) .Statements associated to default and followed cases executed until a break statements comes.    When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.      The following rules apply to a switch statement:   * The variable used in a switch statement can only be integers, convertable integers (byte, short, char), strings and enums. * The value for a case must be the same data type as the variable and it must be a constant or a literal. * Same value cannot repeat. * Break is optional * Default is optional and maximum one. * Default should appear at the end of the switch, but it’s not mandatory. * Each case is followed by the value to be compared to and a colon. | |
| **Array** |
|  |
| What is Array? |
| Array stores a fixed-size sequential collection of elements of the same type.  The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1. |
| How to declare and create array? |
| Declaring and creating Array in two steps:  Step1 : Declare Array   1. dataType[] arrayRefVar; // preferred way. 2. dataType arrayRefVar[]; // works but not preferred way.   Step2 : Create Array   1. arrayRefVar = new dataType[arraySize];   Declaring and creating Array in one steps:   1. dataType[] arrayRefVar = new dataType[arraySize]; // When values are unknown 2. dataType[] arrayRefVar = new dataType[value0, value1, ..., valuek]; // When values are known |
| Write program to print an array?  Write program to get maximum value of an array?  Write program to get sum of all elements of an array? |
|  |
| What java.util.Arrays class ? |
| The java.util.Arrays class contains below popular methods:   |  | | --- | | **Method & Description in java.util.arrays class** | | [**static <T> List<T> asList(T... a)**](https://www.tutorialspoint.com/java/util/arrays_aslist.htm)  This method returns a fixed-size list backed by the specified array. | | [**static int binarySearch(int[] a, int key)**](https://www.tutorialspoint.com/java/util/arrays_binarysearch_int.htm)  This method searches the specified array of ints for the specified value using the binary search algorithm. The array must be sorted before making this call.If it is not sorted, the results are undefined. | | [**static int binarySearch(Object[] a, Object key)**](https://www.tutorialspoint.com/java/util/arrays_binarysearch_object.htm)  This method searches the specified array for the specified object using the binary search algorithm. | | [**static int[] copyOf(int[] original, int newLength)**](https://www.tutorialspoint.com/java/util/arrays_copyof_int.htm)  This method copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. | | [**static int[] copyOfRange(int[] original, int from, int to)**](https://www.tutorialspoint.com/java/util/arrays_copyofrange_int.htm)  This method copies the specified range of the specified array into a new array. | | [**static boolean deepEquals(Object[] a1, Object[] a2)**](https://www.tutorialspoint.com/java/util/arrays_deepequals.htm)  This method returns true if the two specified arrays are deeply equal to one another. | | [**static int deepHashCode(Object[] a)**](https://www.tutorialspoint.com/java/util/arrays_deephashcode.htm)  This method returns a hash code based on the "deep contents" of the specified array. | | [**static String deepToString(Object[] a)**](https://www.tutorialspoint.com/java/util/arrays_deeptostring.htm)  This method returns a string representation of the "deep contents" of the specified array. | | [**static boolean equals(int[] a, int[] a2)**](https://www.tutorialspoint.com/java/util/arrays_equals_int.htm)  This method returns true if the two specified arrays of ints are equal to one another. | | [**static void fill(int[] a, int val)**](https://www.tutorialspoint.com/java/util/arrays_fill_int.htm)  This method assigns the specified int value to each element of the specified array of ints. | | [**static void sort(int[] a)**](https://www.tutorialspoint.com/java/util/arrays_sort_int.htm)  This method sorts the specified array of ints into ascending numerical order. | | [**static String toString(int[] a)**](https://www.tutorialspoint.com/java/util/arrays_tostring_int.htm)  This method returns a string representation of the contents of the specified array of ints. | |
| **Date/Time – old** |
|  |
| What are the classes Date Time API available in Java 7? |
| Java 7 has the following date and time classes and methods.  System.currentTimeMillis()  java.util.Date  java.sql.Date  java.sql.Timestamp  java.util.Calendar  java.util.GregorianCalendar  java.util.TimeZone  java.text.SimpleDateFormat |
| What is the use of System.currentTimeMillis() ? |
| System.currentTimeMillis() is A static method that returns the current date and time as milliseconds since January 1st 1970. |
| What is the use of java.util.Date ? |
| Java's java.util.Date class represents date and time  Today most of the methods in the class are deprecated in favor of the java.util.Calendar class.  You can still use the java.util.Date class to represent a date though.  It has two constructor :  Date() : To initializes the object with the current date and time.  Date(long millisec) : This constructor accepts an argument that equals the number of milliseconds that have elapsed since midnight, January 1, 1970. |
| What are the methods java.util.Date supports? |
| |  |  | | --- | --- | |  |  | | boolean after(Date date) | Returns true if the invoking Date object is later than the one specified by date | | boolean before(Date date) | Returns true if the invoking Date object is earlier than the one specified by date | | Object clone( ) | Duplicates the invoking Date object. | | int compareTo(Date date) | Compares the value of the invoking object with that of date. Returns 0 if the values are equal.  Returns a negative value if the invoking object is earlier than date.  Returns a positive value if the invoking object is later than date. | | int compareTo(Object obj) | Operates identically to compareTo(Date) if obj is of class Date. Otherwise, it throws a ClassCastException. | | boolean equals(Object date) | Returns true if the invoking Date object contains the same time and date as the one specified by date | | long getTime( ) | Returns the number of milliseconds that have elapsed since January 1, 1970. | | int hashCode( ) | Returns a hash code for the invoking object. | | void setTime(long time) | Sets the time and date as specified by time, which represents an elapsed time in milliseconds from midnight, January 1, 1970 | | String toString( ) | Converts the invoking Date object into a string and returns the result. | |
| What is the use of java.text.DateFormat class ? |
| There are two classes for formatting date in java:  DateFormat  SimpleDateFormat.  The DateFormat class is an abstract class. Provides various methods to format and parse date and time in java in language independent manner.  java.text.Format is the parent class  java.text.SimpleDateFormat is the subclass |
| List down some of the methods of DateFormat class ? |
| |  |  | | --- | --- | | Public Method | Description | | Date parse(String source)  throws ParseException | converts string into Date object. | | final String format(Date date) | converts given Date object into string. | | static final DateFormat getTimeInstance() | returns time formatter with default formatting style for the default locale. | | static final DateFormat getDateInstance() | returns date formatter with default formatting style for the default locale. | | static final DateFormat getDateTimeInstance() | returns date/time formatter with default formatting style for the default locale. | | static final DateFormat getInstance() | returns date/time formatter with short formatting style for date and time. | | static Locale[] getAvailableLocales() | returns an array of available locales. | |
| What does java.text.SimpleDateFormat class does? |
| The java.text.SimpleDateFormat class provides methods to format and parse date and time in java.  The SimpleDateFormat is a concrete class which inherits java.text.DateFormat class. |
| What is java.util.GregorianCalendar class does ? |
| The java.util.GregorianCalendar class is a concrete subclass of Calendar abstract class and provides the standard calendar system used by most of the world.  It is a hybrid calendar that supports both the Julian and Gregorian calendar systems.  The Julian calendar specifies leap years every four years, whereas the Gregorian calendar omits century years which are not divisible by 400.  Field: Following are the fields for java.util.GregorianCalendar class −  static int AD − This is the value of the ERA field indicating the common era (Anno Domini), also known as CE.  static int BC − This is the value of the ERA field indicating the period before the common era (before Christ), also known as BCE.  List of popular constructors :  GregorianCalendar():  This constructs a default GregorianCalendar using the current time in the default time zone with the default locale.  GregorianCalendar(int year, int month, int dayOfMonth):  This constructs a GregorianCalendar with the given date set in the default time zone with the default locale.  GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute, int second):  This constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.  GregorianCalendar(TimeZone zone, Locale aLocale):  This constructs a GregorianCalendar based on the current time in the given time zone with the given locale. |
| List out some popular methods of GregorianCalendar class ? |
| |  | | --- | | [**void add(int field, int amount)**](https://www.tutorialspoint.com/java/util/gregoriancalendar_add.htm)  This method adds the specified (signed) amount of time to the given calendar field, based on the calendar's rules. | | [**int getActualMaximum(int field)**](https://www.tutorialspoint.com/java/util/gregoriancalendar_getactualmaximum.htm)  This method returns the maximum value that this calendar field could have, taking into consideration the given time value and the current values of the getFirstDayOfWeek, getMinimalDaysInFirstWeek, getGregorianChange and getTimeZone methods. | | [**int getActualMinimum(int field)**](https://www.tutorialspoint.com/java/util/gregoriancalendar_getactualminimum.htm)  This method returns the minimum value that this calendar field could have, taking into consideration the given time value and the current values of the getFirstDayOfWeek, getMinimalDaysInFirstWeek, getGregorianChange and getTimeZone methods. | | [**boolean isLeapYear(int year)**](https://www.tutorialspoint.com/java/util/gregoriancalendar_isleapyear.htm)  This method determines if the given year is a leap year. | | [**void roll(int field, int amount)**](https://www.tutorialspoint.com/java/util/gregoriancalendar_roll_amount.htm)  This method adds a signed amount to the specified calendar field without changing larger fields. | | [**setTimeZone(TimeZone zone)**](https://www.tutorialspoint.com/java/util/gregoriancalendar_settimezone.htm)  This method sets the time zone with the given time zone value. | |
| What does java.util.TimeZone do ? |
| Java TimeZone class represents a time zone offset, and also figures out daylight savings. It inherits the Object class.     |  |  | | --- | --- | | Method | Description | | static String[] getAvailableIDs() | It is used to get all the available IDs supported. | | static TimeZone getDefault() | It is used to get the default TimeZone for this host. | | String getDisplayName() | It is used to return a name of this time zone suitable for presentation to the user in the default locale. | | String getID() | It is used to get the ID of this time zone | | int getOffset(long date) | It is used to return the offset of this time zone from UTC at the specified date. | | void setID(String ID) | It is used to set the time zone ID | |
| **Regular Expression** |
|  |
| Which pakage contains Regular Expressions related methods? |
| * A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. * Java provides the java.util.regex package for pattern matching with regular expressions. * The java.util.regex package primarily consists of the following three classes −  1. Pattern Class –   Pattern class provides no public constructors.  To create a pattern, you must first invoke one of its public static compile() methods, which will then return a Pattern object.   1. Matcher Class –   Like the Pattern class, Matcher defines no public constructors. You obtain a Matcher object by invoking the matcher() method on a Pattern object.  A Matcher object is the engine that interprets the pattern and performs match operations against an input string.   1. PatternSyntaxException –   A PatternSyntaxException object is an unchecked exception that indicates a syntax error in a regular expression pattern. |
| List some of the Subexpression ? |
| |  |  | | --- | --- | | **Subexpression** | **Matches** | | **^** | **Matches the beginning of the line.** | | **$** | **Matches the end of the line.** | | **.** | **Matches any single character except newline. Using m option allows it to match the newline as well.** | | **[...]** | **Matches any single character in brackets.** | | **[^...]** | **Matches any single character not in brackets.** | | \A | Beginning of the entire string. | | \z | End of the entire string. | | \Z | End of the entire string except allowable final line terminator. | | **re\*** | **Matches 0 or more occurrences of the preceding expression.** | | **re+** | **Matches 1 or more of the previous thing.** | | **re?** | **Matches 0 or 1 occurrence of the preceding expression.** | | **re{ n}** | **Matches exactly n number of occurrences of the preceding expression.** | | **re{ n,}** | **Matches n or more occurrences of the preceding expression.** | | **re{ n, m}** | **Matches at least n and at most m occurrences of the preceding expression.** | | a| b | Matches either a or b. | | (re) | Groups regular expressions and remembers the matched text. | | (?: re) | Groups regular expressions without remembering the matched text. | | (?> re) | Matches the independent pattern without backtracking. | | **\w** | **Matches the word characters.** | | **\W** | **Matches the nonword characters.** | | **\s** | **Matches the whitespace. Equivalent to [\t\n\r\f].** | | **\S** | **Matches the nonwhitespace.** | | **\d** | **Matches the digits. Equivalent to [0-9].** | | **\D** | **Matches the nondigits.** | | \A | Matches the beginning of the string. | | \n | Back-reference to capture group number "n". | | \b | Matches the word boundaries when outside the brackets. Matches the backspace (0x08) when inside the brackets. | | \B | Matches the nonword boundaries. | | \n, \t, etc. | Matches newlines, carriage returns, tabs, etc. | | \Q | Escape (quote) all characters up to \E. | | \E | Ends quoting begun with \Q. | |
| Methods of the Matcher Class |
| 1. Index Methods : Index methods provide useful index values that show precisely where the match was found in the input string        1. Study Methods: Study methods review the input string and return a Boolean indicating whether or not the pattern is found          1. Replacement Methods :Replacement methods are useful methods for replacing text in an input string |
| List PatternSyntaxException Class Methods ? |
| * A PatternSyntaxException is an unchecked exception that indicates a syntax error in a regular expression pattern. * The PatternSyntaxException class provides the following methods to help you determine what went wrong |
| **Methods and Constructors** |
|  |
| Java method ? |
| * A Java method is a collection of statements that are grouped together to perform an operation.Methods represent the behaviour of object. * Create a method :      * Calling a method :  1. The methods returning void:   System.out.println("This is tutorialspoint.com!");   1. The method returning value:   int result = sum(6, 9); |
| What is void keyword ? |
| When a methods has no return value , then it’s return type is written as void. |
| Passing parameter by value ? |
| * Here we just pass the value of parameter to a method instead of passing actual parameter. * The values of the arguments remains the same even after the method invocation. |
| Method Overloading |
| * When a class has two or more methods by the same name but different parameters, it is known as method overloading. * It is different from overriding. In overriding, a method has the same method name, type, number of parameters, etc. |
| What is constructor ? |
| * A constructor initializes an object when it is created. A constructor to give initial values to the instance variables defined by the class, or to perform any other startup procedures required to create a fully formed object. * Constructor has the same name as its class and is syntactically similar to a method. * However, constructors have no explicit return type. * All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to default value. * However, once you define your own constructor, the default constructor is no longer used. |
| What is Parameterized Constructor? |
| * A constructor that accepts one or more parameters. * Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name. |
| The this keyword ? |
| * ‘this’ is a keyword is used as a reference to the curent object. Hence using ‘this’ you can refer the members of a class such as constructors, variables and methods.      * Important − The keyword this is used only within instance methods or constructors. * Usage of ‘this’ keyword :  1. Differentiate the instance variables from local variables 2. Call one type of constructor (parametrized constructor or default) from other constructor or method. It is known as explicit constructor invocation. 3. Call method from other constructor or method. |
| Variable Arguments(var-args) ? |
| * JDK 1.5 enables you to pass a variable number of arguments of the same type to a method. * The parameter in the method is declared as follows.In the method declaration, you specify the type followed by an ellipsis (...).      * Only one variable-length parameter may be specified in a method and this parameter must be the last parameter. Any regular parameters must precede it. |
| The finalize( ) Method? |
| * It is possible to define a method that will be called just before an object's final destruction by the garbage collector. * This method is called finalize( ) and it can be used to ensure that an object terminates cleanly. For example, you might use finalize( ) to make sure that an open file owned by that object is closed. * To add a finalizer to a class, you simply define the finalize( ) method. Here, the keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class.      * The Java runtime calls that method whenever it is about to recycle an object of that class. * You cannot know when or even if finalize( ) will be executed. For example, if your program ends before garbage collection occurs, finalize( ) will not execute. |
| **Nested Class** |
|  |
| Nested Classes |
| * A class too can have another class as its member. * The class written within is called the nested class * The class that holds the inner class is called the outer class. |
| Types of Nested classes ? |
| * Nested classes are divided into two types –  1. Static nested classes − These are the static members of a class. 2. Non-static nested classes − These are the non-static members of a class. |
| What is Static Nested Class? |
| * A nested class which is a static member of the outer class is called Static Nested class. * It can be accessed without instantiating the outer class, using other static members. * Important: Just like static members, a static nested class does not have access to the instance variables and methods of the outer class. * Inner classes are a security mechanism in Java. We know a class cannot be associated with the access modifier private, but if we have the class as a member of other class, then the inner class can be made private. And this is also used to access the private members of a class.       Or |
| Types of Non Static Inner classes? |
| * Non Static Inner classes are of three types depending on how and where you define them.  1. Inner Class 2. Method-local Inner Class 3. Anonymous Inner Class |
| What is Inner Class (Non static)? |
| * Write a non static class within a class. * Unlike a class, an inner class can be private. * Inner classes are also used to access the private members of a class.     Or |
| What is Method-local Inner Class ? |
| * we can write a class within a method and * This class will be a local type you can not add modifer with the class. Like local variables, the scope of the inner class is restricted within the method. * A method-local inner class can be instantiated only within the method where the inner class is defined. |
| What is Anonymous Inner Class? |
| * An inner class declared without a class name is known as an anonymous inner class. * In case of anonymous inner classes, we declare and instantiate them at the same time. * Generally, they are used whenever you need to override the method of a class or an interface. |
| How can you use Anonymous Inner Class as Argument? |
| * Generally, we can implement the interface, extend the abstract class, and pass the object to the method. But in all the three cases, you can pass an anonymous inner class to the method directly. |
| **Inheritance - (Single , Multilevel, Hierarchical)** |
|  |
| What is inheritance ? |
| * Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. * With the use of inheritance the information is made reusable, manageable in a hierarchical order. * extends is the keyword used to inherit the properties of a class. |
| What is superclass ,base class, parent class? |
| The class whose properties are inherited is known as superclass (base class, parent class). |
| What is subclass , derived class, child class? |
| The class which inherits the properties of other is known as subclass (derived class, child class) |
| Types of Inheritance? |
|  |
| What is the use of super keyword? |
| * The super keyword is opposit to this keyword.This point to self class’s member , super point to super class’s member. * Following are the scenarios where the super keyword is used.  1. To differentiate the members of superclass from the members of subclass, if they have same names.      1. It is used to invoke the superclass constructor from subclass.     Note: Subclass’s default constructor calls Superclass’s default constructor automatically.If you write super(); in subclass default constructor, it will be ignored.  Hence    Equivalent to |
| What is the use of instanceof Keyword ? |
| * The instanceof operator to check determine is an object belongs to particular class or not. * An object may directly be part of associate with a class or associate indirectly. |
| What is Overriding |
| * overriding means to override the functionality of an existing method. * If a class inherits a method from its superclass, then there is a chance to override the method provided that it is not marked final. * in the runtime, JVM figures out the object type and would run the method that belongs to that particular object. |
| Rules for Method Overriding? |
| * The argument list should be exactly the same as that of the overridden method. * The return type should be the same or a subtype of the return type declared in the original overridden method(in the superclass). * The access level cannot be more restrictive than the overridden method's access level.   For example: If the superclass method is declared public then the overridding method in the sub class cannot be either private or protected.   * A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final. * Instance methods can be overridden only if they are inherited by the subclass. * A method declared final cannot be overridden. * A method declared static cannot be overridden but can be re-declared. * Constructors cannot be overridden. * If a method cannot be inherited, then it cannot be overridden. |
| **Polymorphism** |
|  |
| What is Polymorphism? |
| A polymorphism can be defined as an object  showing different behaviours at different stages of its life cycle.or  performing single action in different ways.  Types of polymorphism :   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Compile Time polymorphism | Compile-Time binding | Static binding | Early binding | Method overloading | It happans in the same class | | Run Time polymorphism | Run-Time binding | Dynamic binding | Late binding | Method overriding | It happans in the different classes | |
| Method Overloading |
| * When a class has two or more methods by the same name but different parameters, it is known as method overloading. * It is different from overriding. In overriding, a method has the same method name, type, number of parameters, etc. |
| What is Method Overriding |
| * overriding means to override the functionality of an existing method. * If a class inherits a method from its superclass, then there is a chance to override the method provided that it is not marked final. * in the runtime, JVM figures out the object type and would run the method that belongs to that particular object. |
| **Abstraction** |
|  |
| What is Virtual Function? |
| * A virtual function is a member function that you expect to be redefined in derived classes. * In Java, all non-static methods are by default "virtual functions." * Only methods marked with the keyword final, which cannot be overridden, along with private methods, which are not inherited, are non-virtual. * In below example, we instantiate two Salary objects.   One using a Salary reference s, and the other using an Employee reference e.   * While invoking s.mailCheck(), the compiler sees mailCheck() in the Salary class at compile time, and the JVM invokes mailCheck() in the Salary class at run time. * However , mailCheck() on e is quite different because e is an Employee reference.   When the compiler sees e.mailCheck(), the compiler sees the mailCheck() method in the Employee class.Here, at compile time, the compiler used mailCheck() in Employee to validate this statement.  At run time, however, the JVM invokes mailCheck() in the Salary class.   * This behavior is referred to as virtual method invocation, and these methods are referred to as virtual methods. |
| What is Abstraction? |
| Abstraction is a process of hiding the implementation details from the user. Оnly the functionality will be provided to the user.  In Java, abstraction is achieved using abstract classes and interfaces.  Advantages of Abstraction  The main benefit of using an abstract class is that it allows you to group several related classes as siblings.  Abstraction helps to reduce the complexity of the design and implementation process of software. |
| Abstract Class |
| A class which contains the abstract keyword in its declaration is known as abstract class.   * Abstract classes may or may not contain abstract methods, i.e., methods without body ( public void get(); ) * But, if a class has at least one abstract method, then the class must be declared abstract. * If a class is declared abstract, it cannot be instantiated. * To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it. * If you inherit an abstract class, you have to provide implementations to all the abstract methods in it. |
| Abstract Methods |
| If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.   * abstract keyword is used to declare the method as abstract. * You have to place the abstract keyword before the method name in the method declaration. * An abstract method contains a method signature, but no method body. * Instead of curly braces, an abstract method will have a semoi colon (;) at the end.   Declaring a method as abstract has two consequences –   * The class containing it must be declared as abstract. * Any class inheriting the current class must either override the abstract method or declare itself as abstract. |
| Example of Abstract Class and method |
|  |
| **Encapsulation** |
|  |
| What is Encapsulation or Datahiding? |
| Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.  Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.  To achieve encapsulation in Java −   1. Declare the variables of a class as private. 2. Provide public setter and getter methods to modify and view the variables values. |
| Difference between Abstraction and Encapsulation? |
|  |
| **Interface** |
|  |
| Difference between Abstract Class and Interface? |
|  |
| What is interface? |
| * An interface contains behaviors that a class implements. To define behaviors , it contains abstract methods(but no need to put abstract keyword for the methods). * Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class. * Along with abstract methods, an interface may also contain  1. Constants 2. default methods(Since Java 8) 3. static methods(Since Java 8) 4. nested types.   Method bodies exist only for default methods and static methods.   * The interface keyword is used to declare an interface. |
| What is the actual use of interface in java? |
|  |
| Similarilty between interface and class |
| * An interface is written in a file with a .java extension, with the name of the interface matching the name of the file. * The byte code of an interface appears in a .class file. * Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name. |
| Difference between interface and class |
| * You cannot instantiate an interface. * An interface does not contain any constructors. * An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final. * An interface can not extended by a class; it is implemented by a class. * An interface can extend multiple interfaces. |
| How to Declaring Interfaces? |
| * The interface keyword is used to declare an interface. * An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface. * Each method in an interface is also implicitly abstract, so the abstract keyword is not needed. * Methods in an interface are implicitly public. |
| How to Implement Interfaces? |
| * A class uses the implements keyword to implement an interface. * Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method. * The signature of the interface method and the same return type or subtype should be maintained when overriding the methods. * An implementation class itself can be abstract and if so, interface methods need not be implemented. * A class can implement more than one interface at a time. * An interface can extend another interface, in a similar way as a class can extend another class. * An interface can extend more than one parent interface. |
| What is Marker or Tagging interface ? |
| An interface with no methods in it is referred to as a tagging interface.  For example, the MouseListener interface in the java.awt.event package extended java.util.EventListener, which is defined as    There are two basic design purposes of tagging interfaces –   1. Creates a common parent − You can use a tagging interface to create a common parent among a group of interfaces.   For example, when an interface extends EventListener, the JVM knows that this particular interface is going to be used in an event delegation scenario.   1. Adds a data type to a class − This situation is where the term, tagging comes from. A class that implements a tagging interface does not need to define any methods (since the interface does not have any), but the class becomes an interface type through polymorphism. |
| Multiple inheritance is not supported through class in java but it is possible by interface, why? |
| 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. |
| What is Default Method in Interface? |
| Java 8 introduces “Default Method” or (Defender methods) new feature, which allows developer to add new methods to the interfaces without breaking the existing implementation of these interface.  It provides flexibility to allow interface define implementation which will use as default in the situation where a concrete class fails to provide an implementation for that method. |
| Can we have static method in interface? |
| Since Java 8, we can have static method in interface. |
| Does Java supports Nested Interface? |
| Yes, An interface can have another interface i.e. known as nested interface. |
| **Package** |
|  |
| What is a Package ? |
| A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations ) providing access protection and namespace management.  It is a good practice to use names of packages with lower case letters to avoid any conflicts with the names of classes and interfaces.  The package statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.  If a package statement is not used then the class, interfaces, enumerations, and annotation types will be placed in the current default package.    A class must then use one of the following techniques for referring to a class in a different package.   1. The fully qualified name of the class can be used.   For example –     1. The package can be imported using the import keyword and the wild card (\*).   For example –     1. The class itself can be imported using the import keyword.   For example –    Note − A class file can contain any number of import statements. The import statements must appear after the package statement and before the class declaration. |
| What is Static Import ? |
| The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name. |
| Java Packages hierarchy ? |
| Java Packages  Java interfaces and classes are grouped into packages. The following lists the java packages, from which you can access interfaces and classes.    java.lang  Package that contains essential Java classes, including numerics, strings, objects, compiler, runtime, security, and threads. This is the only package that is automatically imported into every Java program.  java.io  Package that provides classes to manage input and output streams to read data from and write data to files, strings, and other sources.  java.util  Package that contains miscellaneous utility classes, including generic data structures, bit sets, time, date, string manipulation, random number generation, system properties, notification, and enumeration of data structures.  java.net  Package that provides classes for network support, including URLs, TCP sockets, UDP sockets, IP addresses, and a binary-to-text converter.  java.awt  Package that provides an integrated set of classes to manage user interface components such as windows, dialog boxes, buttons, checkboxes, lists, menus, scrollbars, and text fields. (AWT = Abstract Window Toolkit)  java.awt.image  Package that provides classes for managing image data, including color models, cropping, color filtering, setting pixel values, and grabbing snapshots.  java.awt.peer  Package that connects AWT components to their platform-specific implementations (such as Motif widgets or Microsoft Windows controls).  java.applet  Package that enables the creation of applets through the Applet class. It also provides several interfaces that connect an applet to its document and to resources for playing audio. |
| **Collections** |
|  |
| What is Collection Framework? |
| The Java Collections Framework is a set of classes and interfaces  These classes and interfaces implement the collection data structures  For example: lists, stack, queue or maps  Almost all collections are derived from the  java.util.Collection interface !!! |
| When and who invented Collection Framework? |
| The collections framework was designed and developed primarily by Joshua Bloch  It was introduced in JDK 1.2 |
| Why should we use Collection Framework? |
| We don’t have to implement every algorithm and data structure from scratch, it has been tested. |
| Can we convert any Collection to Array? |
| ~ toArray() method can transform any collection into  a one-dimensional array |
| Can we use for each loop instead of Iterator for Colllection ? |
| the Collection interface extends the java.lang.Iterable interface  This is why we can use for-each loop !!! |
| Please provide basic example of Iterator? |
|  |
| Please provide basic example of for each in Collection? |
|  |
| Please provide hyrarchy of Collection Framework ? |
| Yelllow : Interface  White : Concreate implentation(Class) which are popular  Dark Gray : Concreate implentation(Class) which are not popular |
| What is ArrayList? |
| * Java ArrayList class uses a dynamic array for storing the elements. * It inherits AbstractList class and implements List interface.        * Memory Allocation For Primitive Type      * Memory Allocation For Object Type Data |
| Important point about ArrayList? |
| * Java ArrayList class can contain duplicate elements. * Java ArrayList class maintains insertion order. * Java ArrayList class is non synchronized. * Java ArrayList allows random access because array works at the index basis. * In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list. |
| ArrayList Constructors? |
|  |
| ArrayList Methods? |
|  |
| ArrayList Sample ? |
|  |
| What is LinkedList? |
| * Java LinkedList class uses doubly linked list to store the elements. * It inherits the AbstractList class and implements List and Deque interfaces.        * Memory Allocation |
| Important points about Java LinkedList ? |
| * Java LinkedList class can contain duplicate elements. * Java LinkedList class maintains insertion order. * Java LinkedList class is non synchronized. * In Java LinkedList class, manipulation is fast because no shifting needs to be occurred. * Java LinkedList class can be used as list, stack or queue. |
| Constructor of LinkedList? |
|  |
| Sample LinkedList ? |
|  |
| What is Vector? |
| * Vector implements List Interface. * Vectors are synchronized:   Like ArrayList it also maintains insertion order but it is rarely used in non-thread environment as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.   * Vectors contain many legacy methods: These methods are not part of the collections framework * Vector uses : new Vector(capacity, capacityIncrement)   The vector's storage increases in chunks the size of capacityIncrement.  We can increase the capacity of a vector before inserting a large number of components  this reduces the amount of incremental reallocation.  SO: every time the vector becomes full --> the size will be incremented by the capacityIncrement |
| Creat a Sample Vector? |
|  |
| What is Stack ? |
| * A Stack is a data structure where you add elements to the "top" of the stack and also remove elements from the top again      * It follows Last In First Out (LIFO) principle * Applications --> for example graph traversing with depth-first search !!! * Stack's are really good for some types of data processing. For instance if you are parsing an XML file using either SAX or StAX * Important methods of Stack :  1. push() method pushes an object onto the top of the Stack. 2. peek() method returns the object at the top of the Stack, but does not remove object 3. pop() method returns the object at the top of the stack, and removes the object from the Stack |
| Sample Stack ? |
|  |
| What is Queue? |
| * We insert the items at the end of the queue and items are removed from the beginning of the queue.      * FIFO structure -> First In First Out !!! * Applications: breadth-first search !!! * Important Methods:  1. add() -> we add items to our queue 2. element() -> approximately the same as peek(): we get the first item without removing it !!! 3. remove() -> this method removes the element at the head of the queue |
| Sample Queue ? |
|  |
| What is Priority Queue? |
| * Based on a priority heap !!!!!! * We assign a priority value to every single items. * The elements of the priority queue are ordered according to their natural ordering defined by the Comparable interface. * Important Methods:  1. add() -> we add items to our queue 2. peek() -> similar to Stack instead of Queue : Retrieves, but does not remove, the head of this queue,or returns null if this queue is empty. 3. poll() -> similar to Stack instead of Queue : Retrieves and removes the head of this queue, or returns null if this queue is empty |
| Sample Priority Queue? |
|  |
| What is Deque: "doubly-ended queue" ? |
| * We can insert and remove from both sides of the queue * it is not thread safe * it allows unlimited insertion of items * ArrayDeque can be used as queue and stack both * When it is used as stack, it is faster than stack * When it is used as queue, it is faster than linkedList !!! * Important Methods :   offerFirst(T t)  offerLast(T t)  removeFirst()  removeLast() |
| Sample Deque ? |
|  |
| What is Hashing? |
| Hashing is a process of converting an object into integer form by using the method hashCode().. A shorter value helps in indexing and faster searches. |
| What is HashMap ? |
| * Java HashMap class implements the map interface by using a hashtable. * It inherits AbstractMap class and implements Map interface.        * A HashMap contains values based on the key. * It contains only unique keys. * It may have one null key and multiple null values. * It maintains no order |
| Please explain internal structure of HashMap? |
| 1. A bucket is one element of HashMap array. It is used to store nodes.   Two or more nodes can have the same bucket. In that case link list structure is used to connect the nodes.   1. Node can represent a class having following objects : 2. int hash 3. K key 4. V value 5. Node next 6. In HashMap, hashCode() is used to calculate the bucket and therefore calculate the index.   *Note : hashCode() method of object class returns the memory reference of object in integer form.*  Insert Data:   1. Initially Empty hashMap: Here, the hashmap is size is taken as 16(default size).   HashMap map = new HashMap();   1. Inserting Key-Value Pair: Putting one key-value pair in above HashMap   map.put(new Key("vishal"), 20);     1. Inserting another Key-Value Pair: Now, putting other pair that is,   map.put(new Key("sachin"),30);       1. In Case of collision: Now, putting another pair that is,   map.put(new Key("vaibhav"),40);    Fetch the data:   1. Fetch the data for key sachin:   map.get(new Key("sachin"));     1. Fetch the data for key vaibahv:   map.get(new Key("vaibhav")); |
| Important point on Hashmap memory allocation ? |
| * Time complexity is almost constant for put and get method until rehashing is not done. * In case of collision, i.e. index of two or more nodes are same, nodes are joined by link list i.e. second node is referenced by first node and third by second and so on. * If key given already exist in HashMap, the value is replaced with new value. * hash code of null key is 0. * When getting an object with its key, the linked list is traversed until the key matches or null is found on next field. * In Java 8 hash elements use balanced trees instead of linked lists after a certain threshold is reached while storing values. This improves the worst case performance from O(n) to O(log n). |
| Constructors of HashMap? |
|  |
| Method of HashMap? |
|  |
| Sample HashMap? |
|  |
| What is LinkedHashmap ? |
| * This implementation differs from HashMap in that it maintains a doubly-linked list running through all of its entries. * This linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (insertion-order). * It inherits HashMap class and implements the Map interface. |
| Implimentation of LinkedHashMap ? |
| * Same as Hashmap , but each Node(Entry) contains , before and after field to keep trach of insertion order. |
|  |
|  |
|  |
|  |