JAVA

* For program alignment in java
* Ctrl+a
* Ctri+i
* Ctrl+shift+f

1. JAVA SE -- Standard Edition-->Standalone application.

2. JAVA ME -- Micro Edition-->small device to develop application.

3. JAVA EE -- Enterprise Edition.

-->It is Platform Independent.

-->write once use anywhere.

-->It's a source file to convert .java extension (src.java)

-->src.java file needs for compile. [The name of the compiler is JAVAC].

-->The JAVAC is the CHECK for rules and syntactical errors. And, then execute the program.

-->After the compilation src.java file is convert to src.class file

-->The .class file will execute, use this commend--->[java Filename].

JAVA--OBJECT ORIENTED PROGRAMMING: OBJECTS:-

1. Objects are elements of a program that has some data which is also known as states.

2. Objects also have behaviours which means they can perform certain operations (Methods).

KEYWORDS

These are the words which having pre-defined meaning used in programming language.

Java is a case sensitive.

Java there are 50 key words. But totally 53 the remaining 3 keywords are literals

IDENTIFIERS:

RULES:

1. Whenever creating an identifier it should an 1st letter alphabet.

2. We cannot create the identifier starting from digit. But after the alphabet n number of digits accept.

3. Identifier are also is case sensitive.

4. Keywords can’t be used as identifiers.

5. Identifiers not allow special character except '\_'and'$'.

6. Can’t be accept space between 2 words.

Once declare the class name public the file name should be same. Otherwise it not compile.

VARIABLES

It's a piece of memory with a given name.

The variable must have contain some data (or) value.

Every variable contain a data type

DATA TYPES

1. Primitive Data Types:

2. Reference Data Types:

1. Primitive Data Types:

(1 byte = 8bit)

1. Byte - [8bits] -> -128 to 127

2. Short - [16 bit] --> -32768 to 32767.

3. Int - [32 bit]

4. Long - [64bit]

5. Float - [32 bit]--7precision

6. Double - [64 bits]--14 precision.

7. Char - [16 bits]

8. Boolean - [true / false] – size is 1 bit.

Control Flow Statements

->if

->if-else

->if-else if

->switch

->if-else

.if we have single line of code no need of writing inside the {} braces.

.but recommended to use inside the {} braces.

.it is possible to write the code without {} braces but it’s not a good practice.

.if we have more than two lines of code we will use {}.

->if-else if (else if ladder):

We go for else if when we have more than one Boolean condition that needs to be satisfy.

Whenever multiple conditions have we go for if-else if.

->switch

* It will only check for the equality.
* Break statement is not mandatory but it is good practice to use.
* Default statement we can write anywhere in the program. After default it is mandatory to use break if we are using the default in anywhere in the program

LOOPING STATEMENTS

When we want to execute same lines of code again and again we use loops.

1. For

for (int initialization; test condition; increment/decrement)

{

Statements.......

Statements.......

}

.initialization will happen only once.

.condition, increment and decrement will happen n number of times according to requirement.

.Inc, Dec we can give along with for loop or with in the for loop.

.if the loop is infinite we can't write any line after the loop. We will get compile time error like unreachable statement.

int i=0;

Ex: for ( ; ;){

system.out.println (i);

i++;

}

System.out.println ("code outside for loop");

}

}

2. While

While (condition)

{

Statements

Statements

}

.first checking the condition and after doing operation.

.in while loop initialization done before the while loop.

.increment and decrement given in while loop.

3. Do-while loop

* Certain lines of program executed after condition check.
* If condition is true it will execute else it will come out of the loop.

**do** {

Statements……

} **while** (condition);

4. ForEach

* Used for iteration of arrays or collection.

METHODS

Methods:

* Methods are named block of code that are related to certain/particular task.

->methods are used for reduce the number of lines of code.

->code reusability.

Method declaration and definition:

Access specifier access modifier return type method name (arg list)

{

//method body

}

->Access specifiers specifies the visibility of any of the code component.

->Access modifiers tells us about component it belongs to a particular class or particular object.

->return type can be any of the primitive data types or it can be references or it can be void also.

If the return type is other than void we have specify return statement at the end of the code. Return statement is always match with the data type.

Method may or may not return any value.

->method name is always start with lowercase with (). Method name exceeding more than one word second word start with uppercase.

->class name always start with uppercase.

-> A method will only execute if it has been called by passing the required arguments.

-> A method which is being called is known as called method. A method which is calling another method is known as callable method.

Arrays

->Array is a group of homogeneous data that has some index and fixed size.

->indexes are used to identify the elements inside the array.

->index of an array will always starts with 0.

->declaration

Int [] array name;

Int [] array name;

Int array name [];

->creation

IntArr = new int [size];

->initialization

IntArr [index] = value;

ArrayIndexOutOfBoundException-whenever we are trying to access the array elements which exceeds size of an array.

Length is the variable that give the size of an array.

**Strings**

**Definition :** String is a sequence of characters.

* String is immutable in nature.
* String is a class in Java.
* Internally string uses character array to store the data.

Declaration and creation of String:

String str = new String (“HELLO”);

String str = “Hello”;

String str;

str = “hello”;

**length():**

* Length() is used to find the length of the string. Return type is integer.

**toCharArray():**

* Converts this string to a new character array. The return type is char
* The length of the newly character array is equal to the string length.

**Syntax:**

String s = “hello”;

char[] ch = s.tochararray();

**charAt (int index):**

* Return type is char.
* Returns the character value at the specified index.
* The length of the character value is 0 to length()-1.

**equals (object obj):**

* Compares this string to specified object.
* Return type is boolean.

**Conditions:**

* If we are giving the input argument should not be null.
* It should be of type string.
* Character sequence should be same as the first string.

**equalsIgnoreCase(object obj):**

* Compare the two strings and returns the Boolean value.
* The length of the two strings must be same. If the length is different the result is false.
* It is case insensitive.

**contains (char ch):**

* Check the specified character is present or not in the given string.
* It is a case-sensitive.
* It will return the Boolean value.

**replace (char arg1, char arg2)**

* It replaces the new string with old string.

**IndexOf (char ch):**

* If the give character is present it will return the index value.
* The return type is integer.
* If the given character is not present in the string it will return -1.

**toUpperCase():**

* It will convert the given string into uppercase.

**toLowerCase():**

* It will convert the given string into lowercase.

**substring (int arg):**

**substring (int arg1, int arg2):**

* It will take two input arguments. Beginning index and the end index.
* The end index is excluded.

**26-08-2019**

**2. Reference type:**

* A reference type is a type that is based on class rather than a primitive data type.
* A reference type can be based on predefined classes in java or classes defined by programmer or developer.

Static and non-static

* Any member of the class that has been declared keyword static is a static member.
* Static members can be invoked by using class name.
* No need of creating the objects to access the static members or methods from one class to another class.
* The members which are created without using static key word is known as non-static members.
* If we want to access non-static members from one class to another class we need to create an object.
* Creating static members inside a method is not allowed in java.

**Local variable and global variable:**

* **Local variable:**
* The variables which are declared inside the scope of the method called as local variable.

**A. primitive type**

**B. reference type**

* Variables can be of two types-local variable and global variable.
* Local variable can be of primitive or it can be of reference type.
* **Global variable:**
* The variables which are created inside the scope of the class called as global variables.

1. **Primitive: static and non-static.**
2. **Reference: static and non-static.**

* Global variable can be of primitive and reference.
* Primitive can be static and non-static.
* Reference can be static and non-static.

**Constructors:**

* Constructors are the special member function of the class which as the same name as that of the class name.
* Whenever we are creating the object of the class constructor is invoked.
* Constructor can have body but Constructor doesn’t have return type.
* If we provide the return type it will act as a normal method. We can call it like as a normal method in java. It will acts like a not-static method.
* If we are not creating a constructor the compiler will create the constructor called as default constructor.
* Constructors are used to initialize the object but not creating an object.
* Constructor can have 0 arguments or ‘n’ number of arguments.

**Types of constructors**:

**1.** Default constructors 🡪 the constructors are created by the compiler.

**2.** User-defined constructors

**3**. Parameterized constructors 🡪 the constructors are created by the user or programmer.

* No argument constructor and default constructor both are different.
* Default constructor doesn’t have body.
* No argument constructor will have body.

**Constructor overloading:**

* Trying to create a same constructor with different arguments.
* Constructor can be differ:
* Type of constructor.
* Number of arguments.
* Order of occurrence of arguments.
* Import is a key word to access the properties from one package to another package.

**Access specifiers:**

* Controls the visibility of the member.

**Public:**

* If the member/ method can be declared as public it can be accessed by any other package or class.
* Visibility is more but security is less.

**Protected:**

* When a method has been declared as protected it can’t be accessed other package.

**Default / package level:**

* When a method declared without any access specifer it means it is being used default or package level. We can’t access outside the package.

**Private:**

* Whena methodhas been declared as private we can’t access outside the class.
* It has lowest visibility and highest security.

**Association:**

* Association in java is Relationship between two different classes**.**
* Can be of different types.

1. one to one.

2. One too many.

3. Many too many.

In java we have two types:

1. Has-a relationship.

A. aggregation

B. composition

2. Is-a relationship.

**Aggregation:**

* Existence of one class is not dependent upon another class.
* E.g. class room is not dependent on trainees or trainees are not dependent on class room.

**Composition:**

* Existence of one class is very much dependent on another class.
* E.g. any organs in human’s body.

**Inheritance**

* The process of acquiring the properties and functions of another class is called as **‘inheritance’.**
* The aim of inheritance is use to provide **re-usability** of code.
* A class whose properties or functionalities are inherited by some other class is called as **parent class** or **super class** or **base class**.
* A class that inherits properties or functionalities from another class is known as **child class** or **sub-class** or **derived class.**
* For inheritance in java we have to make use of **extends** keyword.
* We can access the properties of both super class and sub class using object of sub class.
* **Final** classes can’t be inherited, final members of a super class can be inherited but can’t be changed.
* **Private** members and constructors of super class can’t be inherited.

Types of inheritance

1. Single Inheritance: one class acquiring the properties of another class.
2. **Multilevel Inheritance :**
3. **Multiple Inheritance:** it is not applicable in java.But it is present in java.

Diamond ring problem.

1. **Hierarchical Inheritance:** when two classes acquiring the properties of single class is called as hierarchical inheritance.
2. **Hybrid Inheritance:** combination of two or more than two inheritances.

Keep a check in hybrid inheritance should not present multiple inheritance**.**

**Super Keyword:**

* Super keyword refers to the object of immediate parent or super class.
* To access the data member of a parent class when both child and parent class have the data member with the same name.

**Super ():**

* Super () method is used to call the super class constructor.
* If we are not passing any arguments through super () it will call the super class no argument constructor.

**Method Overriding:**

* Declaring a method in child class which has already been declared in parent class is known as method overriding.
* Method overriding is done to provide implementation specific to a child class.
* Overriding method present in child class. Whereas overridden method is present in parent class.
* Private, Static and Final methods can’t be override.
* Should not change the method declaration while overriding.

**Advantages:**

* We can provide the implementation in the child class method without changing the code present in the parent class.

**Method Overloading:**

* Return type, access specifiers are not affect on method overloading.
* Writing multiple methods with same type of arguments, number of arguments, and order of occurrence of the methods.
* MO is a feature in java that allows to have same methods (sane name) in a single class more than once. Provided the arg list differ
* Number of parameters.
* Order of parameters.
* Data type of parameters.

WAP that performs basic arithmetic operations having different argument list?

**Polymorphism:**

* An object showing the different stages in its different life is called as polymorphism.
* Poly -> many, morph -> forms.
* Method overriding is a type of polymorphism, it is the example for run time polymorphism.

**There are two types of polymorphism:**

**1. Compile time polymorphism: ex-method overloading**

**2. Runtime polymorphism: ex-method overriding.**

**1. Compile time polymorphism:**

* Binding the method declaration with method definition at the time of compilation by the compiler is called as compile time polymorphism.
* Example: method overloading.
* It is also now as early binding or static binding.

**2. Runtime polymorphism:**

* Binding the method declaration with method definition by JVM by runtime is known as runtime polymorphism.
* Example: method overriding.
* Real-time example Gmail-compose, outbox, inbox etc.
* ATM m/c – amount balance, amount withdraw, etc.
* It is also called as late binding or dynamic binding.

**ABSTRACTION**

**Definition:**

* Hiding the implementation details and providing the functionality is called as abstraction.
* Example ATM m/c, system.out.println(), mobile phone and applications inside the phone.
* Real- time example main () inside the program, scientific calculator.

**Why abstraction necessary**?

* Whenever we develop an application we are sending the application in the form of .class file. We will not send the actual file (.java file).

**How to achieve abstraction?**

1. Abstract class

2. Interface

**Abstract class:**

* Any class which is declared with keyword abstract is called as abstract class.
* Any method that has been declared with a keyword abstract is called as abstract method.
* Abstract method doesn’t have implementation/definition/body.
* Abstract classes can have both abstract methods as well as concrete methods.
* Any class having an abstract method should be declared with a keyword abstract.
* Abstract class can have concrete methods.
* If a class has been declared as abstract then that class should be extended by any child class.
* For abstract class we can’t create objects. But an abstract class can have constructor.
* With the help of abstract class we can have 0 to 100% abstraction.

**Interface**

Definition: its java definition block by default data members are final and static, methods are public and abstract.

* Interfaces can’t have constructors.
* Interface can extends more than one interface.

**Difference between interface and Abstract class:**

**Abstract class:**

* The class which is declared with a keyword abstract is called as abstract class.
* In abstract class if we want to provide the definition in concrete class we use extends keyword.
* In abstract class constructors are allowed.
* Class cant extends interface.
* If we want to write concrete method inside the interface we have to declare that method with default or static.
* Class cant extends two abstract classes at a time.

Interface:

* In interface by default methods are abstract and public, data members are static and final.
* In interface if we want to provide the implementation in the subclass (implementation class) we use implements keyword.
* In interface constructors are not allowed.
* Class can implements two or more than two interfaces.
* In interface till 1.7v concrete methods are not allowed.
* In interface initialization is mandatory.

**Types of interfaces**

**1. Marker interface:**

* The interface that doesn’t have any method inside it is called as marker interface.

Ex: Cloneable –java.lang package, Remote, serializable.

.

**2. Functional interface**:

* It should have only one abstract methods.
* @Functional Interface ensures that it’s a functional interface.

**3. Typical interface-**

* It contains more than one abstract method.
* If we want to make typical interface to functional interface we have to use @Functional Interface annotation.

**Encapsulation**

**Definition**: Encapsulation is a mechanism with which we wrap-up the data members and function members into a single unit is called as encapsulation.

* Encapsulation is achieved by private access specifier.
* If we want to access or modify the private data from another class we use public methods called getters and setters.
* Getters methods are used to access the private data members from another class.
* Setter’s methods are used to manipulate the data members.

**Packages:**

* Packages are the collection of classes and interfaces**.**
* If the classes present in different packages, and we want to access data we need to import that package.
* If we have same class in different packages then we have to give full qualified class name to create the object.

**Final keyword:**

* If the variable declared with final we can’t reinitialize.
* When we are declaring variable using final keyword we have to use capital letter.
* Final methods can’t be override.
* Final classes cannot be inherited, but final class can acquire/inherit the properties from super class.
* Final class can’t be a super-class, final class can be sub-class.

**Object Class**

* In java each and every class directly or indirectly inherits the properties of object class.
* Object class is a super most class in java.
* Each and every class either pre-defined class or a user-defined class is a child class of object.

**Methods of object class**

**1. getClass()**

**2. Clone()**

**3.toString()**

**4.equals()**

**5.hashCode()**

**6.wait()**

**7. wait(long)**

**8. wait (long, int)**

**9. notify()**

**10. notifyAll()**

**11. finalize ()**

**Strings**

* Strings are reference type, not value types, such as int or Boolean.
* As a result, a string variableholds a reference to an object created from the class, not value of the string itself.
* Even though strings aren’t primitive types, the java compiler has some features designed to let you work with strings almost as though they were primitive types.
* String is the only class where operator overloading is supported in java.
* String is immutable. An immutable object is an object that once created can never be changed.
* String pool contains two parts

1. Constant pool

2. Non-constant pool

* Within the constant pool duplicates are not allowed whereas within non-constant pool duplicates are allowed.

**String Builder and String Buffer**

* The string is powerful class, but it’s not very efficient.
* Because string objects are immutable, any method of the string class that modifies the string in any way must create new string object.
* To overcome this problem java offers two alternatives to the string class**: string** **builder and string buffer classes**.
* The **StringBuilder** and **StringBuffer** classes are mirror images. Both have the same method and performs the same string manipulation.
* The only difference is that **stringBuffer** class is thread safe whereas **stringBuilder** not a thread safe.
* Constructs a string buffer with no characters in it and an initial capacity of a capacity of 16 characters.

1. public StringBuffer()

{

Super (16);

}

2. public StringBuffer(int capacity)

{

Super (capacity);

3. Constructs a string buffer initialized to the contents of the specified string. The initial capacity of the string buffer is {@code 16} + length of the string argument.

Public StringBuffer (String str)

{

Super (16)

}

* Public StringBuffer(CharSequence seq)

{

this(seq.length() + 16);

append (seq);

}

* **StringBuilder Constructors**

1. public StringBuilder ()

{

Super(16);

}

1. Public StringBuilder(int capacity)

{

Super(capacity);

}

1. Public StringBuilder(String seq)()
2. {
3. This.(seq.length)
4. }

**Difference between String, StringBuffer, StringBuilder**

**String**

* Immutable
* Not thread safe
* Metods are not synchronized.
* Object will be created in two ways with or without new keyword.
* + operator is supported.
* hashcode (), toString(), equals() of object class are overridden.

**StringBuffer ()**

* Mutable.
* Thread safe.
* Methods are synchronized.
* Object can be created with new keyword.
* + operator is not supported.
* Only toString() of object class is overridden.

**StringBuilder ()**

* Mutable.
* Not thread safe.
* Methods are synchronized.
* Object can be created with new keyword.
* + operator is not supported.
* Only toString() of object class is overridden.
* Little more efficient than stringBuffer.

**EXCEPTION**

* Exception is an error event that can happen during the execution of a program and disturbs the normal flow of execution.
* Exception in java can arise from different kind of situations such as wrong data entered by user , hardware failure, network connection failure, database server down etc.
* Whenever an error occurs while executing a statement creates an exception object and the normal flow of the program halts and jre tries to find someone that can handle the raised exception.
* The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurs, type of exception etc.
* The process of creating the exception object and handling it over to runtime environment is called as “throwing the exception”.
* If appropriate exception handler is found, exception object passed to the handler to process it. The handler is said to be “catching the exception”.
* Java exception handling is a framework that is used to handle runtime errors only, compile time errors are not handled by exception handling in java.
* Java exceptions are hierarchical and inheritance is used to categorize different types of exceptions.
* Throwable is the parent class of java hierarchy and it has two child objects.

1. Error

2. Exception.

Exception Handlers:

1. try

2. catch

3. throw

4. throws

5. finally

**Types of Exception**

1. Checked Exception:
2. Unchecked Exception:

**Difference between exception and errors in java**?

**Exceptions:**

* We can recover the exception by either using try-catch block or throwing exceptions back to caller.
* Exceptions include both checked as well as unchecked type.
* Program itself is responsible for causing exceptions.
* All exceptions occurs at run time but checked exceptions are known to compiler while unchecked are not.
* They are defined in java.lang.Exception package.

Ex: checked Exception: SQLException, IOExceptionUnchecked Exceptions: AIOBE

**Errors:**

* Recovering from error is not possible.
* All errors in java are unchecked type.
* Errors are mostly caused by the environment in which program is running.
* They are defined in java.lang.Error package.
* Errors occurs at run time not known to the compiler.

Ex: java .lang.StackOverflowError, java.lang.OutOfMemoryError.

**Collections**

* Collections are like containers that group multiple items into single unit.
* Collections are used almost every programming language and when java arrived, it also came with few collection classes, Vector, stack, Hash table, Array.
* Java 1.2 provides collection framework that is architecture to represent and manipulate collections in java in a standard way.
* Java collection framework consists following parts:

1. Interfaces
2. Implementation classes.

* Java collection framework interfaces provides abstract data type to represent collection.
* Java.util.collection is the root of the collection framework.
* Java.util.list, java.util.set, java.util.queue.
* Implementation classes:ArrayList, LinedList, HashMap, TreeMap, HashSet.
* All the core collection interfaces are generic.
* The <E> syntax is for generics and when we declare collection we should use it to specify the type of Object it can contain.
* It helps to reduce the run-time errors by type-checking the objects at compile-time.
* It throws two exceptions:

1. ClasscasteException

1. UnSupportedOperatorRxception.

**Collection Interface:**

* This is the root of the collection hierarchy.
* A collection represents a group of elements into a single entity.
* The java platform doesn’t provide any direct implementations of this interfaces.

**Methods**

1. add() - to add the elements into the collection.
2. remove() – to remove
3. size()
4. isEmpty()
5. iterator()
6. contains()
7. containsAll()
8. addAll()
9. removeAll()
10. listIterator()

**Set Interface:**

* A collection that contains no duplicate elements.
* At most one null value.
* Implementation classes - HashSet, Tree Set, and LinkedHashSet.
* It implements Serializable and Cloneable but not Random access.
* Set interface doesn’t allow random-access to an element in the collection.
* We can use iterator or for each loop to traverse the elements in set.

HashSet:

* This class implements the set interface supported by HashTable.
* It makes no guarantees as to the iteration order of the set.
* This class permits null elements.
* Initial capacity is 16 and load factor is .75 or 75%.

**Constructors:**

1. public HashSet(){

map = new HashMap();

}

1. public HashSet(collection<? Extends E> c){

map = new HashMap<>(Math.max((int) (c.size()/.75f) + 1, 16));

addAll(c);

}

1. public HashSet(int initialCapacity, float loadFactor){

map=new HashMap<>(initialCapacity, loadFactor);

}

1. public HashSet(int initialCapacity) {

map = new HashSet(initialCapacity);

}

1. HashSet(int initialCapacity, float loadFactor, Boolean dummy) {

map = LinedHashMap<>(initialCapacity, loadFactor)

}

Constructs a new empty linked hash set.

**List Interface:**

* An ordered collection of elements, that allows duplicate elements.
* It allows multiple null elements.
* It implements Serializable, Cloneable, and RandomAccess.
* Implementation classes: ArrayList and LinedList.

**Queue Interface:**

* A collection designed for holding elements prior to processing.
* Beside basic collection operations queue provides additional insertion, extraction and inspection operations.
* Additional methods:

1. boolean offer(E e) – inserts the specified element into this queue it is possible to do so immediately without violating capacity restrictions.

\* It will return if the element is added into the queue else false.

\* throws classcasteException if the class of the specified elements

2. poll ():

\* retrieves but does not remove the head of this element.

\*this method differs from peek only in that it throws an exception called NoSuchElementException if queue is empty

1. element():

\*retrieve the element but does not remove head of the queue.

\*it returns null if queue is empty.

4. peek ():

**Map (I)**

* An object that maps keys to value.
* A map can’t contain duplicate keys. Each key can map to at most one value.

1. HashMap.

2. TreeMap.

3. LinkedHashMap.

Methods:

1. put () – to add the elements into the map.

2. get () – retrieve the elements from the map.

3. contains-key () – returns the key set

4 .contains-value () – returns the value set.

5. size () -

6. isEmpty ()

**Benefits of CF:**

* Reduces the development effort.
* Increased quality.
* Reusability and Interoperability.

**TreeSet**

* TreeSet(Navigable Map<E, Object> m){

this.m=m;

}

* public TreeSet(){

this(new Tree Map<E, Object>())

}

Constructs a new tree empty set sorted according to the natural sorting order.

It will use comparable interface.

* Public TreeSet (comparator<? Super E> comparator){

this(new Tree Map<>(comparator))

}

Constructs a new empty tree set sorted according to the specified comparator.

All elements inserted into the set must be mutually comparable.

Comparator.compare(e1, e2) if both are same type then no exception

If we are comparing two different types then add method will throw classcasteException.

* Public TreeSet (collection<? Extends E> c){

this();

addAll(c);

}

* Public TreeSet(SortedSet<E> s){

this (s.comparator());

addAll ();

}

**ArrayLst:**

* Resizable array implementation of list interface.
* public ArrayList(int initialCapacity)

{

}

It will throw an exception says that IllegalArgumentException. When the initial capacity is < 0.

* Public ArrayList(){

}

Default capacity is 10.

* Public ArrayList(Collection c)

{

}

This throw NullPointerException when collection is empty.

Methods:

* public void trimToSize() {

}

Helps us to manipulate the size of the array.

It helps to trim the memory which is unused.

* Public void ensureCapacity(int minCapacity) {

}

It increases the arrayList capacity.

**Java Lambda Expressions**

* Lambda expression is a new and important feature of java which was included in Java SE 8.
* It provides the clear way to represent functional expression using an expression.
* It helps to iterate, filter, and extract data from collection.
* The lambda expression is used to provide the implementation of a functional interface.
* It saves lot of code.
* In case of lambda expression, we don’t need to define a method again for providing the implementation. We just write implementation code.
* Java lambda expression is treated as a function, so java compiler doesn’t create .class file.

Functional Interface:

* Lambda expression provides implementation for functional interface.
* An interface which has only one abstract method is known as functional interface.

**Threads**

* A thread is a sequence of executable code within a larger program.
* The main thread that starts automatically when we run the program.

Ex: downloading files and listening songs at a time.

* Two or more threads are running at a time is called as multi-threading.

**Thread Creation:**

1. By extending thread class.

2. By implementing Runnable interface.

* Thread class and Runnable interface both are present in java.lang package.
* The class overrides the run() which is present in Thread class.
* Every thread will have three important properties:

1. **Thread Name**: thread name can be created by programmer in order to identify the threds.

We can set and get the name for thread by using setName() and getName().

1. **Thread Id**: created and assigned by thread scheduler to every single thread in order to identify them uniquely.

We can’t set the id but we can get the id by using getId().

1. **Thread Priority**: it is used by thread scheduler to decide the order of execution of the given thread.

The priority of the thread is an integer value range between 1 – 10. We can set the priority of the thread using setPriority ().

We can get the priority by using getPriority ();

Max-priority: 10

Min-Priority: 1.

Normal-Priority: 5.

If we give the priority <1 and > 10 we will get illegalArgumentException.

* Two or more threads are using one resource it leads to race condition it leads the data inconsistency to overcome this problem we use synchronized keyword.

**Thread Life Cycle:**

* NEW – A thread that has not yet started is in this state.
* RUNNABLE – A thread executing in the java virtual machine is in this state {start()}.
* BLOCED – A thread that is blocked waiting for a monitor lock in this state {yield (), sleep (), join ()}.
* WAITING – A thread that is waiting indefinitely for another thread to perform a particular action is in this state {wait (), notify (), notifyAll ()}.
* TERMINATED – A thread that has exited is in this state{stop()}.

**Race Condition:**

* Two or more threads accessing one resource at a time it will create race condition.

**Synchronization:**

* In many cases concurrently running threads share data and two threads try to do operations on the same variable at the same time. This often results in corrupted data as two thread try to operate on the same data.
* Coding technique it will help you to overcome race condition.
* The popular solution to provide some kind of lock primitive. Only one thread can acquire a particular lock at any particular time. This can be achieved by using a keyword “synchronized”.

**Deadlock:**

* Deadlock in java is a programming situation where two or more threads are blocked forever.
* To overcome deadlock situation we use ITC-inter thread communication. it can be achieved by wait(), notify(), notifyAll().

**Questions:**

1 difference between run () and start ().

2 difference between sleep () and wait ().

Start()

* When a program call start() a new thread is created then the run() method is executed.
* Can’t invoked more than one time otherwise throws IllegalStateException.
* Defined in java.lang.Thread class.

Run()

* If we call the run() the no new thread will be created and run() will be executed as a normal method call on the current calling thread itself.
* Multiple invocations is possible.
* Defined in java.lang.Runnable interface.

Wait()

* Used to pause the execution of a particular thread.

**RegEx(Regular Expression):-**

* The regular expression in java defines a pattern for String.
* Regular expression can be used to search, edit or manipulate text.
* A re is not a language specific but they differ slightly for each language.
* Java Regex class are present in java.util.regex package.
* In regex ‘\_’ is consider as a character not a special character.
* Contains 3 classes

**1 Pattern –**

Pattern object is the compiled version of the regular expression. Pattern class doesn’t have any public constructor and we use its public static method compile () to create the pattern object by passing regular expression argument.

**2 Matcher –** Matcher is a java regex engine object that matches the input String pattern object created.

Matcher doesn’t have any public constructor and we get a matcher object using pattern object matcher () it takes the input String as argument.

**3 PatternSyntaxException**