**Prinsip Dasar Java**

***JAVA FUNDAMENTAL***

1. **Garis Besar /*Overview***
2. **Keuntungan /*Advantage***

Keuntungan kunci dari pembelajaran pemrograman bahasa Java:

*The key advantages of learning Java Programming*

1. Orientasi Objek */ Object Oriented*

Di Java, semuanya adalah objek. Java dapat dengan mudah berkembang sejak berdasarkan model objek.

*In Java, everything is an Object. Java can be easily extended since it is based on Object model.*

1. Rencana kerja berdiri sendiri */ Platform Independent*

Kode byte didistribusikan ke jejaring web dan diterjemahkan oleh *Java Virtual Machine* (JVM) dimanapun rencana kerja berjalan.

*This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.*

1. Sederhana / *Simple*

Jika kamu mengerti konsep dasar PBO (Pemrograman Berorientasi Objek) Java, akan lebih mudah untuk menguasainya.

*If you understand the basic concept of OOP (Object Oriented Programming) Java, it would be easy to master.*

1. Aman / *Secure*

Dapat mengembangkan anti-virus, anti-perusak sistem. Ototentikasi tekhnik berdasarkan kunci umum enkripsi.

*Enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.*

1. Netral Arsitektur / *Architecture-neutral*

Java kompilasi menghasilkan sebuah arsitektur netral objek format berkas, dimana membuat kode yang terkompilasi dapat di eksekusi banyak prosesor, dengan kehadiran Java runtime system.

*Java compile generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.*

1. Portabel / *Portable*

Netral arsitektur dan tidak mempunya implementasi ketergantungan aspek dari spesifikasi membuat Java pportabel.

*Architecture-neutral and having no implementation dependent aspect of the specification makes Java portable.*

1. Ketahanan */ Robust*

Java mengupayakan untuk mengakhiri situasi rawan kesalahan dengan menekankan terutama pada pengecekan kesalahan waktu kompilasi dan pengecekan saat waktu berjalan.

*Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.*

1. **Tipe Aplikasi Java */ Types Of Java Application***

Berikut 4 tipe dasar aplikasi yang dapat dibuat dengan pemrograman java.

*There are mainly 4 types of application that can be created on java programming*

1. Aplikasi Mandiri */ Standalone Application*

Aplikasi mandiri dikenal juga desktop aplikasi atau berbasis windows aplikasi. Sebagai contoh adalah pemutar media, anti-virus, dll. AWT dan Swing digunakan di aplikasi mandiri Java.

*Standalone application are also known desktop Application or windows-based applications. Examples are media player, antivirus, etc. AWT and Swing are used in Java standalone application.*

1. Aplikasi Jaringan */ Web Application*

Berjalan di sisi server dan membuat halaman dinamis disebut juga aplikasi jaringan. Sebagai contoh adalah Servlet, JSP, Struts, Spring, Hibernate, JSF, dsbnya.

*Run on server side and create dynamic page is called a web application. For example is Servlet, JSP, Struts, Spring, Hibernate, JSF, etc.*

1. Aplikasi perusahaan */ Enterprise Application*

Sebuah aplikasi di distribusikan secara alami, seperti aplikasi bank, dsbnya. Dengan keuntungan seperti keamanan tingkat tinggi, memuat keseimbangan, dan pengelompokan. EJB digunakan untuk membuatnya.

*An application is distributed in nature, such as banking applications, etc. It has advantages like high-level security, load balancing, and clustering. EJB is used for creating that.*

1. Aplikasi Seluler / *Mobile Application*

Dibuat untuk perangkat seluler, menggunakan Android dan Java ME.

*Created for mobile device, Android and Java ME are used for that.*

1. **Edisi / Rencana Kerja Java / *Java Platform / Editions***

Terdapat rencana kerja atau edisi pada Java:

*There are 4 platforms or editions of Java:*

1. Java SE (Edisi Standar) / *Java SE (Standard Edition)*

Termasuk pemrograman API java seperti java.lang, java.io, java.net, java.util, java.sql, java.math, dsbnya. Termasuk inti topic seperti PBO (Pemrograman Berorientasi Objek, String, Regex, Pengecualian, Inner class, Multithreading, I/O Stream, Jaringan, AWT, Swing, Reflection, Collection, dsbnya.

*It includes Java programming API such as java.lang, java.io, java.net, java.util, java.sql, java.math, etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.*

1. Java EE (Edisi Perusahaan) / *Java EE (Enterprise Edition)*

Biasanya digunakan untuk mengembangkan web dan aplikasi perusahaan. Termasuk topic seperti Servlet, JSP, Web Services, EJB, JPA, dsbnya.

*Mainly used to develop web and enterprise applications. It includes topics like Servlet, JSP, Web Services, EJB, JPA ,etc.*

1. Java ME (Edisi Mikro) / *Java ME (Micro Edition)*

Panggung mikro untuk aplikasi yg berjalan.

*Micro platform for mobile applications.*

1. Java FX / *Java FX*

Digunakan untuk mengembangkan aplikasi kaya, ringan pengguna antarmuka API.

*Used to develop rich applications, lightweight user interface API.*

1. **Dasar Java / *Java Basic***
2. **Variable Java / *Java Variable***
3. Variabel Lokal / *Local Variable*

Variabel didefinisikan di dalam metode, konstruktor, blok disebut juga variabel local. Deklarasi dan inisialisasi didalam metode dan variable akan dihancurkan ketika metode selesai.

*Variable define inside methods, constructors, blocks are called a local variable. Declare and initialize within the method and the variable will be destroyed when the method has completed.*

1. Variabel Contoh / *Instance Variable*

Variable contoh adalah variable didalam kelas tetapi diluar suatu metode. Variable ini terinisialisasi ketika class terinisialisasi, dapat di akses dari dalam suatu metode, konstruktor, atau blok dari kelas khusus.

*Instance variables are variable within a class but outside any method. These variable are initialized when the class is instantiated, can be accessed from inside any method, constructor, or blocks of that particular class.*

1. Variabel Kelas / *Class Variable*

Variable dideklarasi didalam suatu kelas, diluar suatu metode dengan kata kunci statis.

*Variable declared within a class, outside any method, with the static keyword.*

1. **Tipe Data Java / *Java Data Types***
2. Tipe Data Primitif / *Primitive Data Types*
3. *byte*
4. *short*
5. *int*
6. *long*
7. *float*
8. *double*
9. *boolean*
10. *char*
11. Tipe Data Objek/ Rujukan / *Reference/Object Data Types*
12. Variabel rujukan dibuat menggunakan konstruktor yg didefinisikan suatu kelas. Digunakan untuk mengakses objek-objek.

*Reference variable are created using defined constructors of the classes. They are used to access objects.*

1. Objek kelas dan varian tipe variable array terdapat dalam rujukan tipe data.

*Class object and various type of array variables come under reference data type.*

1. Nilai bawaan variabel adalah null.

*Default value variable is null.*

1. Digunakan untuk merujuk suatu objek tipe yang dideklarasikan atau suatu tipe yang kompatibel.

*Used to refer any object of the declared type or any compatible type.*

1. Java Literal / *Java Literals*

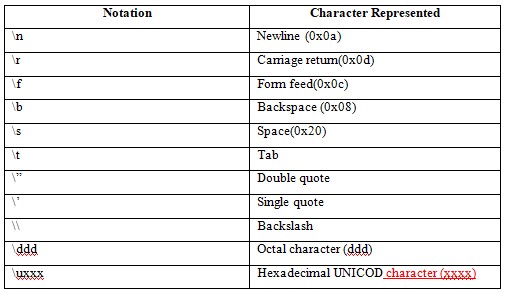
**

Table 1.1 Java Literal

1. **Objek dan Kelas / *Object And Class***
2. Objek */ Object*

Sebuah objek dapat didefinisikan sebagai contoh dari kelas. Suatu kesatuan yg mempunyai perlakuan dan bentuk diketahui sebagai objek.

*An Object can be defined as an instance of a class. Any entity that has behaviour and state is known as an object.*

1. Kelas / *Class*

Kumpulan objek disebut juga sebagai kelas. Sebuah kelas dapat juga didefinisikan sebagai cetak biru darimana kamu dapat membuat objek individu.

*Collection of objects is called a class. A class can also be defined as a blueprint from which you can create an individual object.*

1. **Konstruktor / *Constructor***
2. Method yang memberikan nilai awal pada saat suatu objek dibuat.

*A Method that will return the initial value when an object is created.*

1. Inisialisasi sebuah objek ketika dibuat.

*Initializes an object when it is created.*

1. Setiap waktu objek dibuat menggunakan kata kunci *new()*, setidaknya satu konstruktor terpanggil.

*Every time an object is created using the new () keyword, at least one constructor is called.*

1. *It has the same name as it is class and is syntactically similar to a method.*
2. *Cannot be abstract, static, final, and synchronized.*
3. *No explicit return type.*
4. *No Argument Constructor (Default constructor)*
5. *Parameterized Constructor*
6. ***Operator***

*There are many types of operators in Java which are given below:*

1. *Unary*

* *Postfix : expr ++, expr --*
* *Prefix : ++expr, --expr, +expr, -expr, ~, !*

1. *Arithmetic*

* *Multiplicative : \*, /, %*
* *Additive : +, -*

1. *Shift*

* *<< (left shift), >> (right shift), >>> (unsigned right shift)*

1. *Relational*

* *Comparison : <, >, <=, >=, instanceof*
* *Equality : ==, !=*

1. *Bitwise*

* *AND : &*
* *Exclusive OR: ^*
* *Inclusive OR: |*

1. *Logical*

* *AND : &&*
* *OR : ||*

1. *Ternary*

* *? :*

1. *Assignment*

* *=, +=, -=, \*=, /==, %=, ^=, |=, <<=,>>=,>>>=*

1. ***Keyword***

*Java key word are also known as reserved words, list are given below:*

*abstract, boolean, break, byte, case, catch, char, class, continue, default, do, double, else, enum, extends, final, finally, float, for, if, implement, import, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throws, transient, try, void, volatile, while.*

1. ***String***

*String is basically an object that represent sequence of char values. The String class is immutable, so that once it is created, a String object cannot be changed. Whenever we change any string, a new instance is created, for mutable string, you can use StringBuffer and StringBuilder. String class implements Serializable, Comparable, and CharSequence interface.*

*There are two ways to create String object:*

1. *String literal*

*String s = “Hello World”;*

1. *By new keyword*

*String s = new String (“Hello World”);*

*Java String class method: charAt, length, format, substring, contains, join, equals, isEmpty, concat, replace, equalsIgnoreCase, split, intern, indexOf, lastIndexOf, toLowerCase, toUpperCase, trim, valueOf, compareTo.*

1. ***Arrays***

*Java provides a data structure, the array, which stores a* ***fixed-size*** *sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of array as a collection of variables of the same type.*

*Arrays class contains various static methods of sorting and searching, comparing, filling arrays element.*

1. *Advantages*

* *Code optimization*

*We can retrieve or sort the data efficiently.*

* *Random access*

*We can get any data located at an index position.*

1. *Disadvantages*

* *Size limit*

*We can store only the fixed size of elements in the array. It doesn’t grow its size at runtime.*

1. *Types Of Arrays*

* *Single dimensional array*
* *Double dimensional array*

1. ***Numbers***

*Normally, when we work with Numbers, we use primitive data types such as byte, int, long, double, etc. However, in development, we come across situations where we need to use objects instead of primitive data types. In order to achieve this, Java provides* ***wrapper classes****. All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of file abstract class Number.*

1. *Boxing*

*Converting primitive data types into object is called boxing.*

1. *Unboxing*

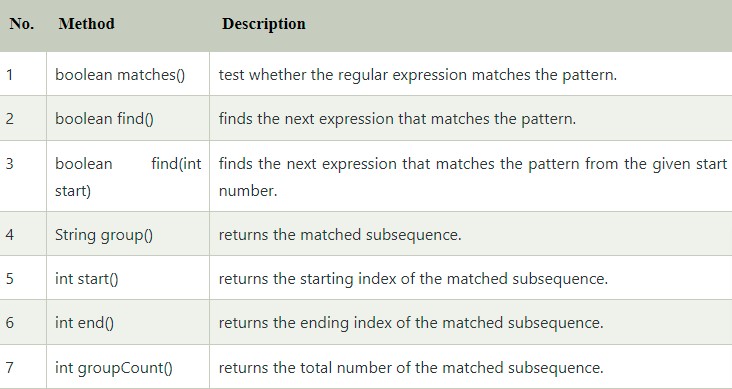
*The wrapper object will be converted back to primitive data type is called unboxing.*

1. ***Regular Expression***

*The java regex or Regular expression is an API to define a pattern for searching or manipulating string*

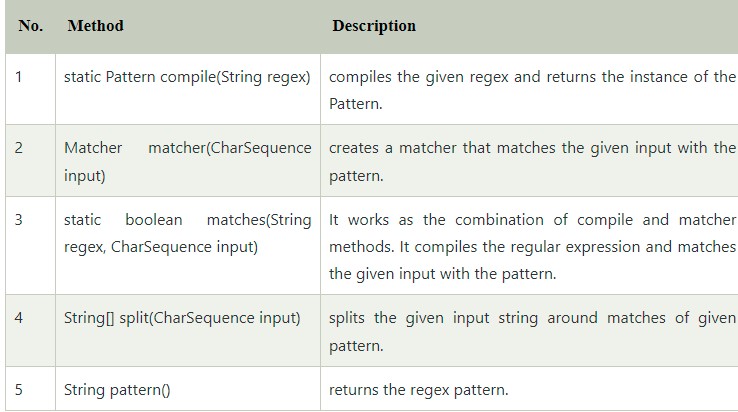
1. *Pattern class*

*It is compiled version of a regular expression. It is used to define a pattern for the regex engine.*

**

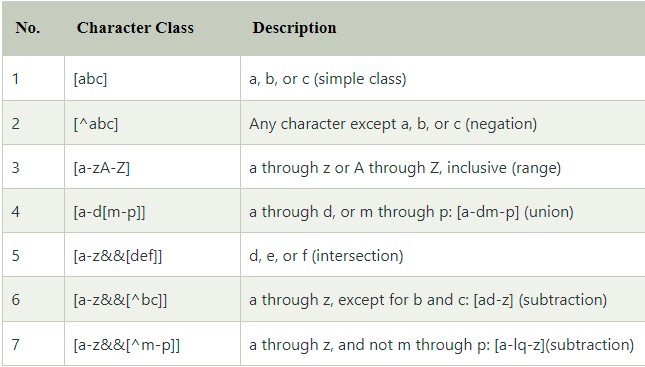
1. *Matcher class*

*It implements the* ***MatchResult*** *interface. It is a regex engine which used to perform match operations on a character sequence.*

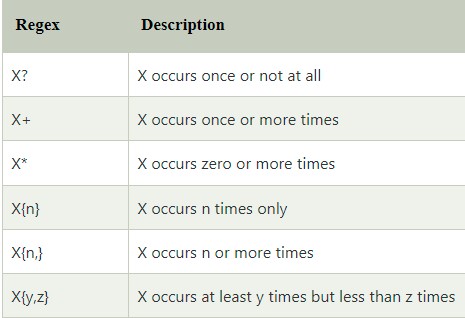
**

1. *Regex character class*

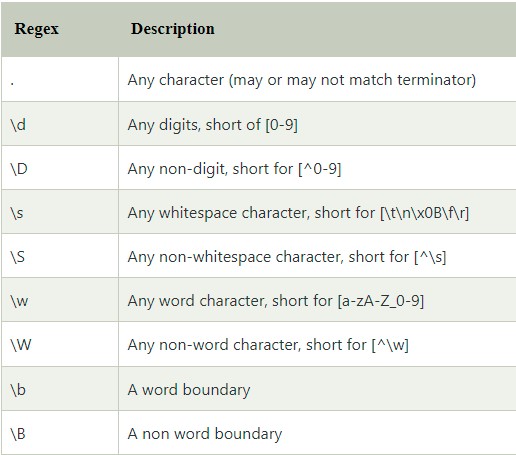
*The . (dot) represent single character.*

**

1. *Regex Quantifier*

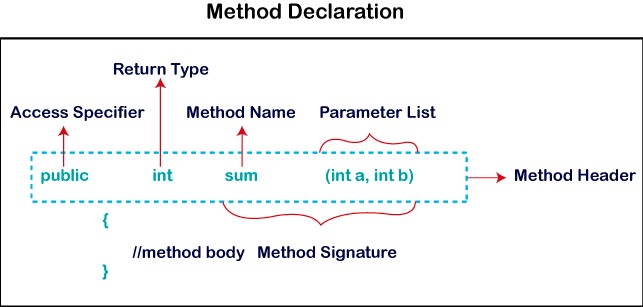
**

1. *Regex Metacharacter*

**

1. *Regular Expression question 1*
2. *Regular Expression question 2*
3. *Java Regex Finder example*
4. ***Method***

*In general, a method is a way to perform some task. Method declaration:*

**

1. *Method Signature*

*It includes the method name and parameter list.*

1. *.Access Specifier*

*Is the access type of the method:*

1. *Public*

*The method is accessible by all classes when we use.*

1. *Private*

*The method is accessible only in the classes in which it is defined.*

1. *Protected*

*The method is accessible within the same package or subclasses in a different package.*

1. *Default*

*It is visible only from the same package only.*

1. *Return Type*

*Return type is a data type that the method returns.*

1. *Method Name*

*It is a unique name that is used to define the name of method.*

1. *Parameter List*

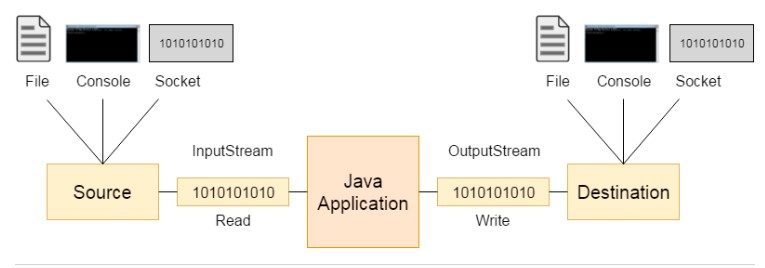
*It is the list of parameters separated by comma and enclosed in the pair of parantheses.*

1. *Method Body*

*It contains all the action to be performed.*

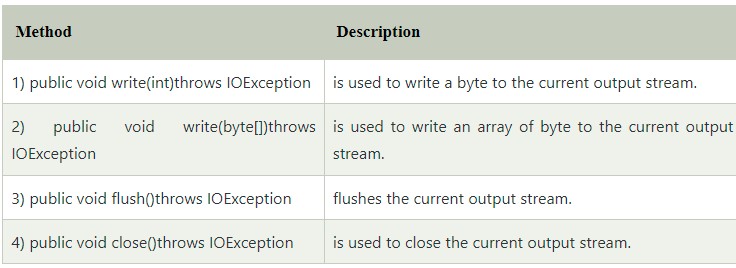
1. ***Files And I/O***

*Java I/O (input/output) is used to process the input and produce the output.*

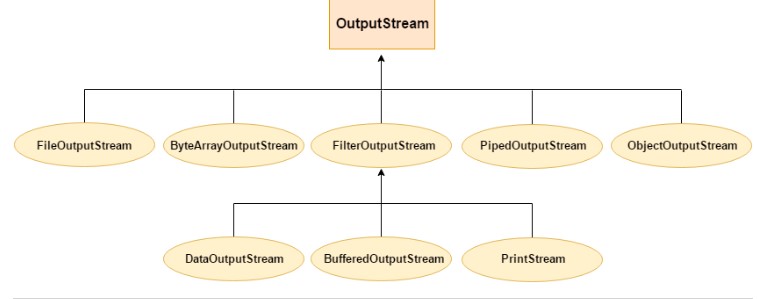
**

1. *Stream*
2. *System.in: standard input stream*
3. *System.out: standard output stream*
4. *System.err: standard error stream*
5. *OutputStream*

*Java application used output stream to write data to destination. It may be a file, an array, peripheral device or socket. Useful method of output stream:*

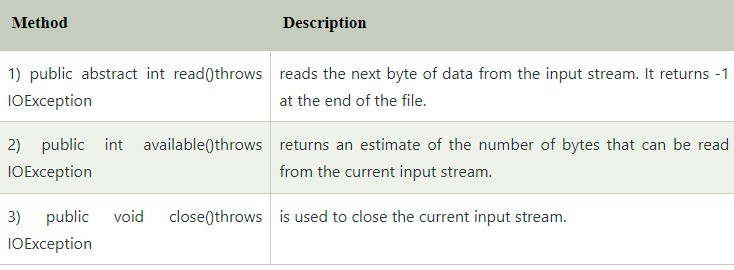
**

*Output stream hierarchy:*

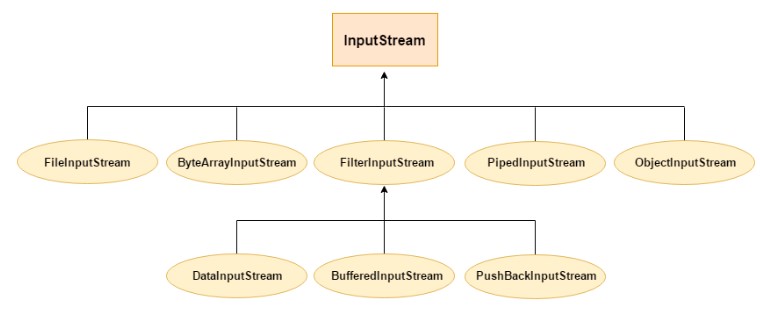
**

1. *InputStream*

*Java application used input stream to read data from a source. Useful method of input stream:*

**

*Input stream hierarchy:*

**

1. ***Multithreading***

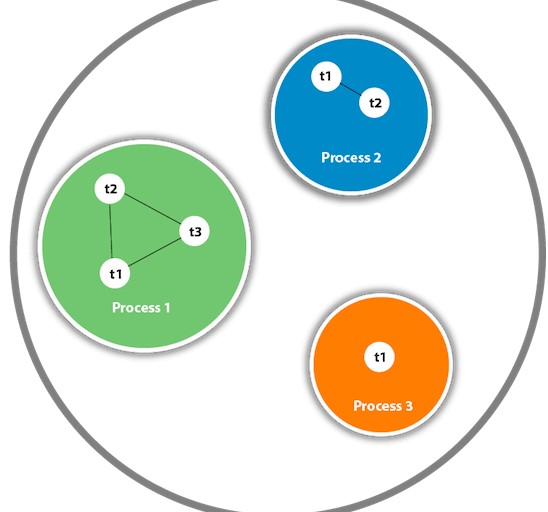
*Multithreading in java is a process of executing multiple threads simultaneously. Advantages of java multithreading:*

1. *It doesn’t block the user because thread is independent and you can perform multiple operations at the same time.*
2. *You can perform many operations together, so it saves time.*
3. *Threads are independent, so it doesn’t affect other threads if an exception occurs in a single thread.*

*Multitasking is a process of executing multiple tasks simultaneously. We use that to utilize the CPU. Multitasking can be achieved by two ways:*

1. *Process Base Multitasking (Multiprocessing)*
2. *Each process has an address in memory. In the other words, each process allocates a separate memory.*
3. *A process is heavyweight.*
4. *Cost of communication between the process is high.*
5. *Switching from one process to another requires some time for saving and loading registers, memory map, updating list, etc.*
6. *Thread Base Multitasking (Multithreading)*
7. *Threads share the same address space.*
8. *A thread is lightweight.*
9. *Cost of communication between the thread is low.*

*A Thread is a lightweight sub process, the smallest unit of processing. Threads are independent. It is a separate path of execution. If there occurs exception in one thread, it doesn’t affect other threads. It uses a shared memory area.*

**

*A thread is executed inside the process. There is context-switching between the threads.*

*Lifecycle of a thread (Thread States):*

1. *New*

*Whenever a new thread is created, it is always in the new state.*

1. *Active*

*When a thread invoke the start() method, it moves from the new state to the active state.*

1. *Runnable:*

*A thread, that is ready to run then moved to the runnable state.*

1. *Running*

*When the threads gets the CPU, it moves from runnable to the running state.*

1. *Blocked / Waiting*

*Whenever a thread is inactive for a span of time (not permanently) then, either the threads is in the blocked state or is in the waiting state.*

1. *Timed Waiting*

*Sometimes waiting for leads to starvation.*

1. *Terminated*

*A thread reaches the termination state because of following reasons:*

1. *When a thread has finished its job, then it exists or terminates normally*
2. *Abnormal termination, it occurs when some unusual events such as an unhandled exception or segmentation fault.*

*A terminated thread means the thread is no more in the system.*

1. ***Exception Handling***

*One of the powerful mechanism to handle the runtime errors so that the normal flow of application can be maintained. Exception is an abnormal condition. Types of java exception:*

1. *Checked Exception*

*The classes that directly inherit the Throwable class except RuntimeException and error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exception are checked at compile-time.*

1. *Unchecked Exception*

*The classes that directly inherit the RuntimeException are known unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutboundException,etc. Unchecked exception are not checked at compile-time, but they are checked at runtime.*

1. *Error*

*Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError, etc.*

*Java provides five keywords that are used to handle the exception: try, catch, finally, throw, throws.*

1. ***Inner classes***

*Java inner class or nested class is a class declared inside the class or interface. Difference between nested class and inner class, an inner class is a part of a nested class. Non static nested class are known as inner class. Advantage inner class:*

1. *Nested class represent a particular type of relationship that it is access all the members (data members and method) of the outer class, including private.*
2. *Nested class are used to develop more readable and maintainable code because it logically group classes or interface in one place only.*
3. *Code optimization. It required less code or write.*

*Types of nested classes:*

1. *Non-static nested class(inner class)*
2. *Member inner class*
3. *Anonymous inner class*
4. *Local inner class*
5. *Static nested class*
6. *Member inner class*

*A class created within class and outside method.*

1. *Anonymous inner class*

*A class created for implementing an interface or extending class. The java compiler decided its name.*

1. *Local inner class*

*A class was created within the method.*

1. *Static nested class*

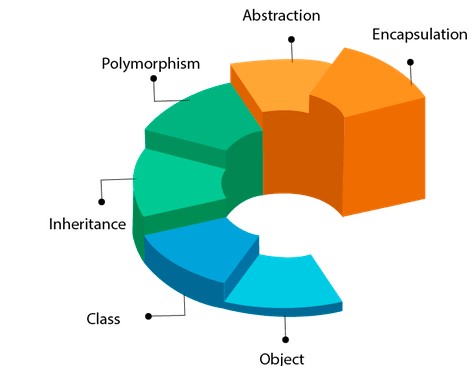
*A static class was created within a class.*

1. *Nested interface*

*An interface created within class interface.*

1. ***Object Oriented Programming (OOP)***

*Object oriented programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concept*

*.*

1. ***Inheritance***

*When one object acquires all the properties and behaviour of a parent object, it is known as inheritance. Why use inheritance in Java:*

1. *Inheritance (IS-A)*

*Inheritance represent the IS-A relationship which is also known as a parent-child relationship. Why use inheritance in Java:*

1. *For method Overriding (so runtime polymorphism can be achieved)*
2. *For code reusability*

*Term used in inheritance:*

1. *Class*

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

1. *Sub class / Child Class*

*Sub class is a class which inherits the other class. It is also called a derived class, extended class, or child class.*

1. *Super class/ Parent Class*

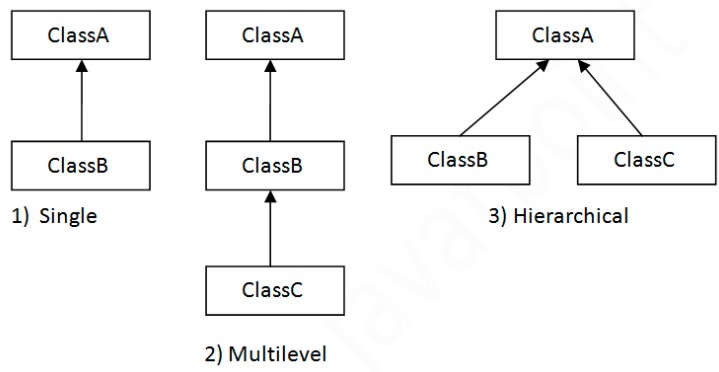
*Super class is the class from where a subclass inherits the features. It is also called a base class or a parent class.*

1. *Reusability*

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

*The* ***extends keyword*** *indicated that you are making a new class that derives from an existing class. The meaning extends is to increase the functionality.*

*Types of inheritance in Java*

**

*Multiple inheritance is not supported in Java through class.*

1. *Aggregation (HAS-A)*

*If a class have an entity reference, it is known as Aggregation. Aggregation represent HAS-A relationship. When use aggregation?*

1. *Code reuse it also best achieved by aggregation when there is no is-a relationship.*
2. *Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.*
3. ***Polymorphism***

*One task is performed in different ways.*

1. *Overloading*

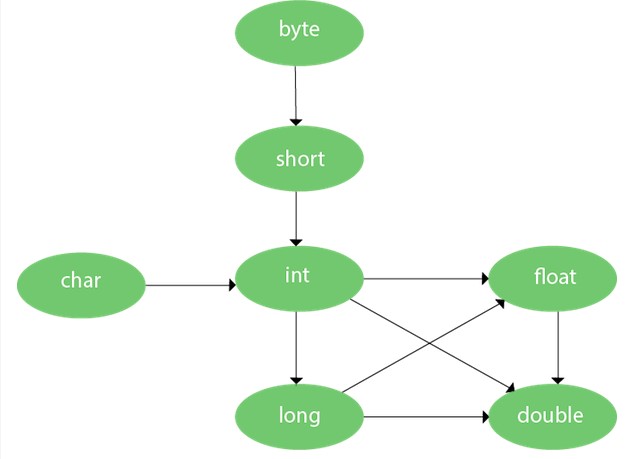
*If a class has multiple methods having same name but different in parameters, it is known as method overloading. Method overloading increases the readability of the program. There are two ways to overload the method in java.*

1. *By changing number of arguments*
2. *By changing the data type*

*In Java, method overloading is not possible by changing the return type of the method only. Compile time error is better than runtime error. So, java compiler renders compiler time error if you declare the same method having same parameters.*

1. *Method overloading and type promotion*

*One type is promoted to another implicitly if no matching data type is found.*

**

*As displayed in the above diagram, byte can be promoted by short, int, long, float or double. If there are matching type arguments in the method, type promotion is not performed. If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity. One type is not de-promoted implicitly for example double cannot be de promoted to any type implicitly.*

1. *Overriding*

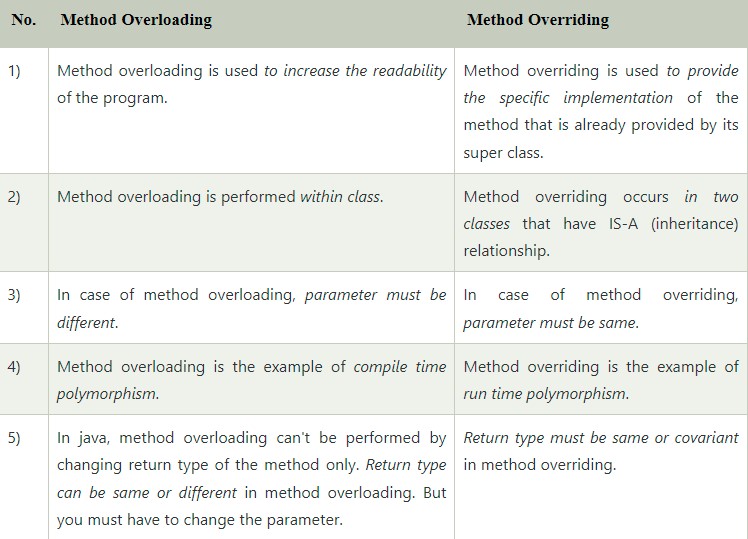
*If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java. Usage of Java method overriding:*

1. *Method overriding is used to provide the specific implementation of a method which is already provided by its super class.*
2. *Method overriding is used for runtime polymorphism.*

*Rules for java method overriding:*

1. *The method must have the same name as in the parent class*
2. *The method must have the same parameters as in the parent class*
3. *There must be an IS-A relationship (inheritance)*

*Java method overriding is mostly used in runtime polymorphism which we will learn in next pages. Static and Main method cannot be overridden. It is because the static method is bound with class where as instance method is bound with an object. Static belongs to class area, and an instance belongs to the heap area.*

**

1. ***Covariant return type***

*The covariant return type specifies that the return type may vary in the same directions as the subclass. Following are the advantages of the covariant return type:*

1. *Covariant return type assist to stay away from the confusing type casts in the class hierarchy and makes the code more usable, readable, and maintainable.*
2. *In the method overriding, the covariant return type provides the liberty to have more to the point return types.*
3. *Covariant return type helps in preventing the run-time ClassCastException on return.*
4. ***Super keyword***

*The super keyword in Java is a reference variable which is used to refer immediate parent class object. Usage of java super keyword:*

1. *Super can be used to refer immediate parent class instance variable*

*We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.*

1. *Super can be used to invoke immediate parent class method*

*The super keyword can also be used to invoke parent class method. It should be used if subclass contains same method as parent class. In other words, it is used if method is overridden.*

1. *Super can be used to invoke immediate parent class constructor*

*The super keyword can also be used to invoke the parent class constructor*

1. ***Instance initialize block***

*Instance initialize block is used to initialize the instance data member. It run each time when object of class is created. Rules for instance initialize block:*

1. *The instance initialize block is created when instance of the class is created.*
2. *The instance initialize block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).*
3. *The instance initialize block comes in the order in which they appear.*
4. ***Final keyword***

*The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:*

1. *Variable*

*If you make any variables as final, you cannot change the value of this variable, because final variable once assigned a value can never be changed.*

1. *Method*

*If you make any method as final, you cannot override it.*

1. *Class*

*If you make any class as final, you cannot extend it.*

1. ***Runtime polymorphism***

*Polymorphism in Java is a concept by which we can perform in single action in different ways. There are two types polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and overriding. If you overload a static method in Java, it is example of compile-time polymorphism.*

***Runtime polymorphism or Dynamic Method Dispatch*** *is a process in which a call to an overridden method is resolved at runtime rather than compile-time. In this process, an overridden method is called through the reference variable of super class. Lets first understand the up casting before runtime polymorphism.*

1. *Up casting*

*If the reference variable of parent class refers to the object of child class, it is known as upcasting. For upcasting, we use the reference variable of class type or an interface type.*

1. *Java runtime polymorphism with data member*

*A method is overridden, not the data members, so runtime polymorphism can’t be achieved by data members.*

1. *Java runtime polymorphism example:*
2. *Java runtime polymorphism with multilevel inheritance:*
3. ***Dynamic binding***

*Connecting a method call to the method body is known as binding.*

1. *Static binding (early binding)*

*When type of the object is determined at compiled time, it is known as static binding.*

1. *Dynamic binding (late binding)*

*When type of the object is determined at run-time, it is known as dynamic binding.*

*Understanding Type:*

1. *Variables have a type*
2. *Reference have a type*
3. *Objects have a type*
4. ***Instanceof operator***

*The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface). The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.*

1. *Down casting with java instanceof operator*

*When subclass type refers to the object of parent class, it is known as downcasting. If we perform it directly, compiler gives Compilation error. If you perform it by typecasting, ClassCastException is thrown at runtime. But if we use instanceof operator, downcasting is possible.*

1. *Down casting without java instanceof operator*
2. ***Abstraction***

*Hiding internal details and showing functionality to the user is known abstraction. Ways to achieve abstraction in java:*

1. ***Abstract class***

*A class which is declared with the abstract keyword is known as an abstract class. It can have abstract and non-abstract methods (method in the body).*

*Points to remember:*

1. *An abstract class must be declared with an abstract keyword*
   1. *It can have abstract and non-abstract method*
   2. *It cannot be instantiated*
   3. *It can have constructors and static method also*
   4. *It can have final methods which will force the subclass not to change the body of the method.*
2. *Abstract method in Java*

*A method which is declared as abstract and does not have implementation is known as an abstract method.*

1. *Abstract class having constructor, data members and method*

*An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main () method. If there is an abstract method in a class, that class must be abstract.*

*Mostly we don’t know about the implementation class (which is hidden to the end user), and an object of the implementation class is provided by the factory method. A factory method is a method that returns the instance of the class. The abstract class can also be used to provide some implementation of the inheritance. In such case, the end user may not be forced to override all the methods of the interface.*

1. ***Interface***

*An interface in Java is a blueprint of a class. It has static constant and abstract method. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. Java interface also represent the IS-A relationship. Why use java interface:*

1. *It is used to achieve abstraction*
2. *By interface, we can support the functionality of multiple inheritance*
3. *It can be used to achieve loose coupling*

*An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all field are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface. The relationship between classes and interface:*

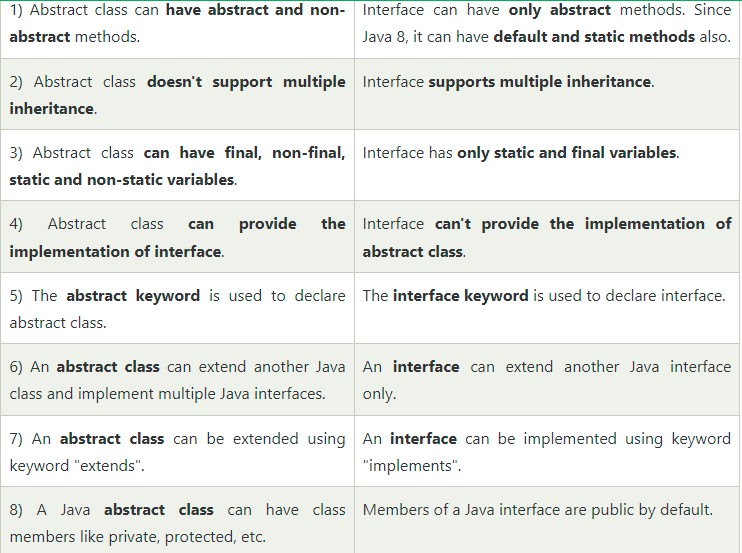
1. *A class extends another class*
2. *An interface extends another interface*
3. *A class implements an interface*

*Multiple inheritance in java by interface, if a class implements multiple interface, or an interface extends multiple interfaces, it is known as multiple inheritance. Multiple inheritance is not supported through class in Java, but it is possible by an interface, because there is no ambiguity. It is because its implementation is provided by the implementation class.*

*Since Java 8, we can have method body, static method in interface. But we need to make it default method.*

*An interface can have another interface which is known as a nested interface.*

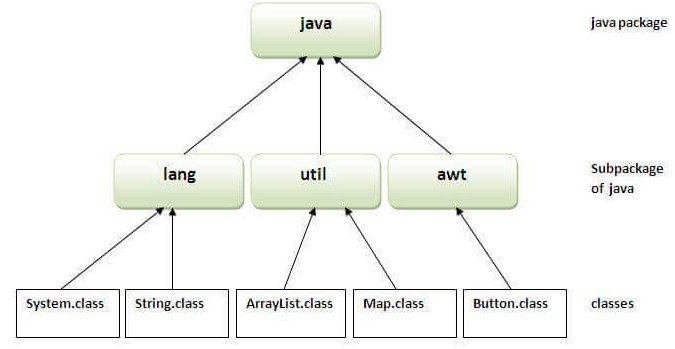
1. ***Abstract versus Interface***

******

1. ***Encapsulation***
2. ***Package***

*Java package is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package. Built-in packages such as java, lang, awt, javax, swing, net, io, util, sql, etc. Advantage of java package:*

1. *Java package is used to categorize the classes and interfaces so that they can be easily maintained.*
2. *Java package provides access protection*
3. *Java package removes naming collision*

**

*How to access package from another package:*

1. *Using packagename.\**

*If you use package.\* then all the classes and interfaces of this package will be accessible but not sub packages. The import keyword is used to make the classes and interface of another package accessible to the current package.*

1. *Using packagename.classname*

*If you import package.classname then only declared class of this package will be accessible.*

1. *Using fully qualified name*

*If you use fully qualified name then only declared class of this package will be accessible. But you need to use fully qualified name every time when you are accessing the class or interface.*

*Package inside the package is called the subpackage. It should be to categorize the package further. The standard of defining package is domain.company.package.*

1. ***Access Modifiers***

*There are two types of modifiers in Java:*

1. *Access modifiers*

*The access modifiers in java specifies the accessibility or scope or a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.*

1. *Private*

*The access level of a private modifier is only within the class.*

1. *Default*

*The access level of a default modifier is only within the package.*

1. *Protected*

*The access level of a protected modifiers is within the package and outside the package through child class.*

1. *Public*

*The access level of public modifiers is everywhere.*

1. *Non-access modifiers*

*There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient, etc.*

1. ***Encapsulation***

*Encapsulation in java is a process of wrapping code and data together into a single unit, for example a capsule which is mixed of several medicines. The Java Bean class is the example of a fully encapsulated class. Advantage encapsulation in Java:*

1. *You can make the class read-only or write-only by providing only a getter or setter method.*
2. *It provides you the control over the data*
3. *It is way to achieve data hiding in Java because other class will not be able to access the data through the private data members.*
4. *The encapsulate class is easy to test*
5. *Easy and fast to create an encapsulated class*
6. ***Object class***

*The object class is the parent class of all the classes in java by default. The object class is beneficial if you want to refer any object whose type you don’t know, notice that parent class reference variable can refer the child class object, known as up casting.*

*The object class provides some common behaviors to all the objects such as object can be compared, cloned, notified, etc.*

1. *Object cloning*

*The object cloning is a way to create exact copy of an object. The advantage of object cloning:*

1. *You don’t need to write lengthy and repetitive codes.*
2. *It is the easiest and most efficient way for copying objects, especially if we are applying it to already developed or an old project.*
3. *Clone () is the fastest way to copy array.*

*Disadvantage of object cloning:*

1. *To use object clone () method, we have to change a lot of syntaxes to our code, like implementing a cloneable interface, defining the clone () method, handling cloneNotSupportException, and finally calling clone().*
2. *We have to implement cloneable interface while it doesn’t have any methods in it.*
3. *Object.clone () is protected, so we have to provide our own clone() and indirect call Object.clone () from it.*
4. *If you want write a clone method in a child class then all of its superclasses should define then clone() method in them or inherit it from another parent class.*
5. *Object.clone () support only shallow copying but we will need to override it if we need deep cloning.*
6. ***Coupling***

*Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. You can use interface for the weaker coupling because there is no concrete implementation.*

1. ***Cohesion***

*Cohesion refers to the level of a component which performs a single well-defined task. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interface.*

1. ***Association***

*Association represent the relationship between the objects. Association can be unidirectional or bidirectional. There can be four types of association between the objects:*

1. *One to one*
2. *One to many*
3. *Many to one*
4. *Many to many*
5. ***Aggregation***

*Aggregation is a way to achieve association. Aggregation represents the relationship where an object contains other object as a part of its state. It represents the weak relationship between objects. It is also termed as a has-a relationship in Java. Like inheritance represent the is-a relationship. It is another way to reuse object.*

1. ***Composition***

*The composition is also a way to achieve association. The composition represent the relationship where one object contains other objects as a part of its state. There is a strong relationship between containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the objects will be deleted automatically.*

1. ***Wrapper Class***

*The wrapper class in java provides the mechanism to convert primitive into object and object into primitive. Where we need to use wrapper class:*

1. *Change the value in method*

*Java support only call by value. So, if we pass a primitive value, it will not change the original value.*

1. *Serialization*

*We need convert the objects into streams to perform the serialization. If we have primitive value, we can convert it in objects through the wrapper class.*

1. *Synchronization*

*Java synchronization works with object in multithreading*

1. *Java.util package*

*The java.util package provides the utility classes to deal with objects*

1. *Collection framework*

*Java collection framework works with objects only*

1. ***Java 8 Feature***

*Java 8 provides following features for Java Programming:*

1. ***Lambda Expressions***

*It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to filter, iterate, and extract data from collection. Saves a lot of code (less coding), don’t need define the method again for providing the implementation, we just write implementation code. Treated function, doesn’t create .class file.*

1. *Functional Interface*

*An interface which has only one abstract method is called functional interface. Java provides an annotation @FunctionalInterface.*

1. *Syntax*

***(argument-list) ->{body}***

*Consisted of three components:*

1. *Argument-list*

*It can be empty or non-empty as well.*

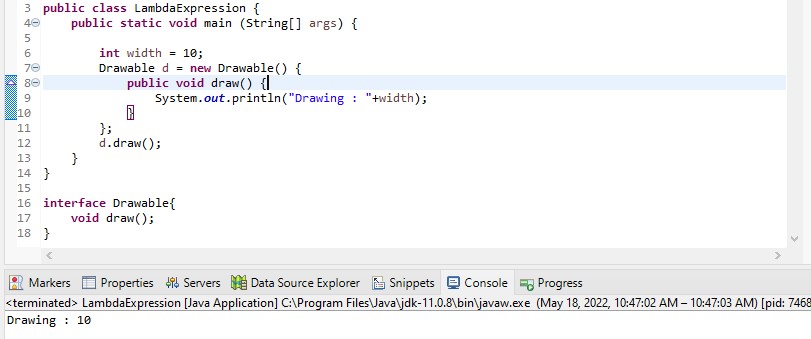
1. *Arrow-token*

*It is used to link arguments-list and body of expression.*

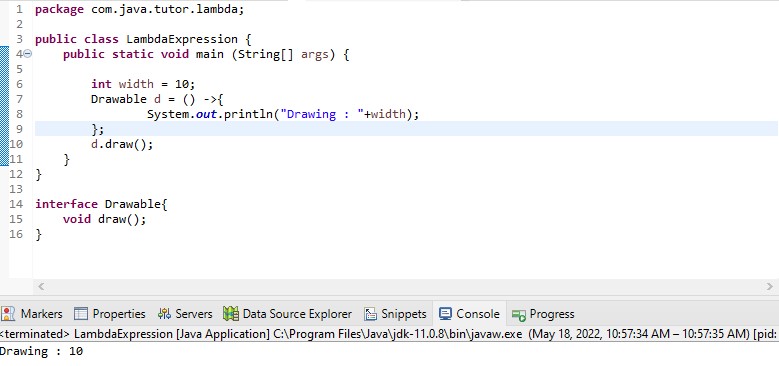
1. *Body*

*It contain expressions and statements for lambda expression.*

*Without Lambda Expression example:*

**

*With Lambda example:*

**

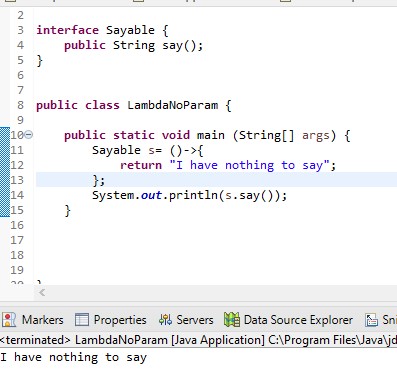
***No parameter syntax***

***() -> {***

***//*** *Body of no parameter lambda*

***}***

*Example:*

**

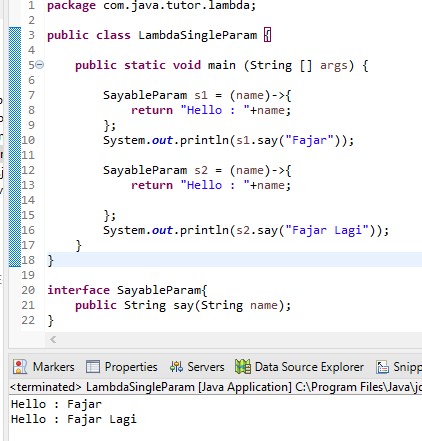
***One parameter syntax***

***(p1) -> {***

***//*** *Body of no parameter lambda*

***}***

*Example****:***

******

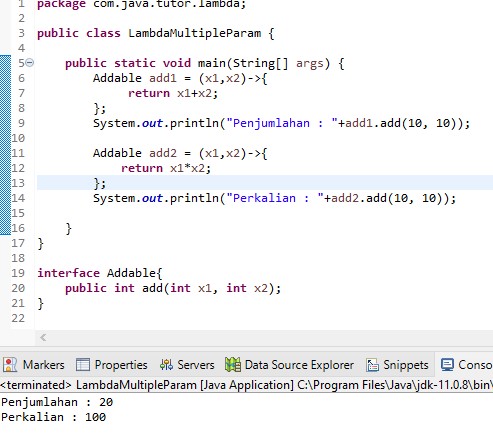
***Two parameter syntax***

***(p1,p2) -> {***

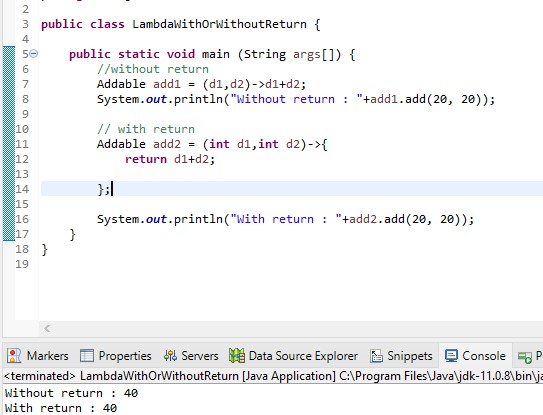
***//*** *Body of no parameter lambda*

***}***

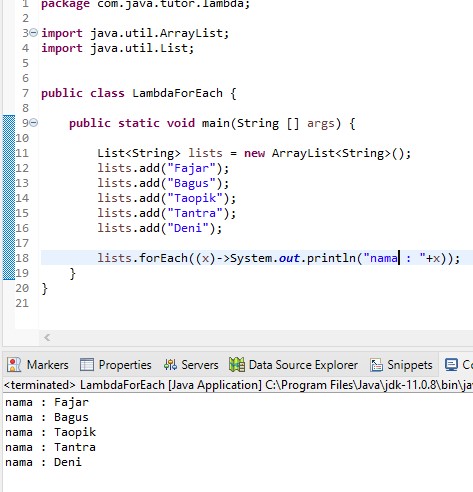
*Example:*

**

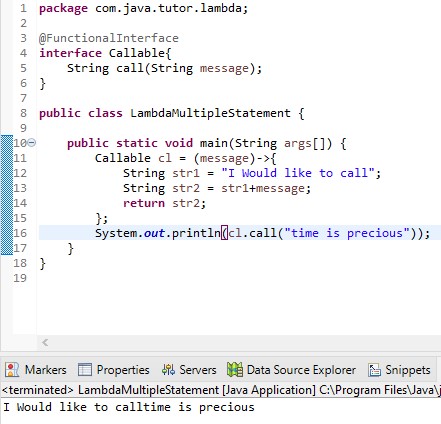
*With or Without return key:*

**

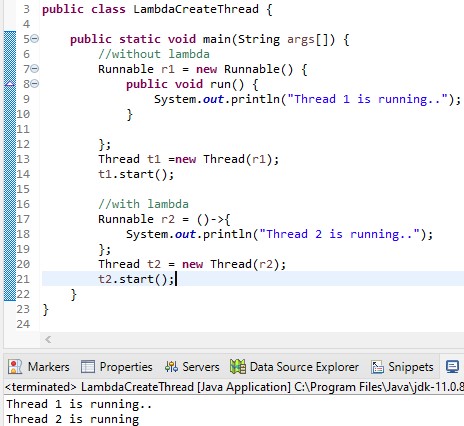
*Foreach ,example:*

**

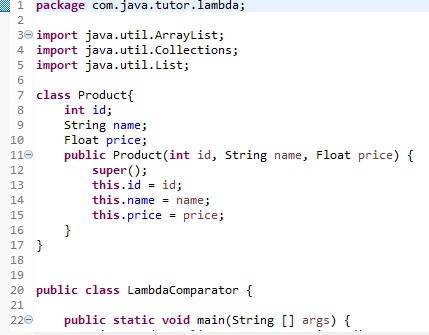
*Multiple statement, example:*

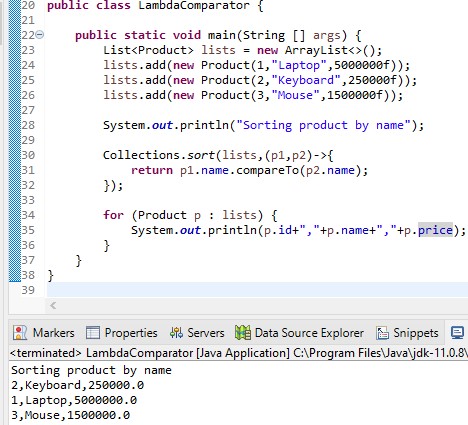
**

*Creating thread,example:*

**

*Comparator, example:*

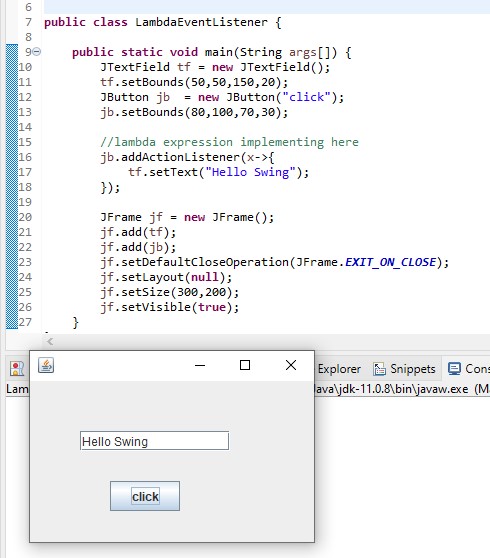
**

**

*Filter collection data, example:*

**

*Event listener, example:*

**

1. ***Method Reference***

*Java provides new feature called method reference in Java 8. Method reference is used to refer method of functional interface. Types of method references:*

1. *Reference to a static method*

*You can refer to static method defined in the class.*

1. *Reference to an instance method*

*Like static method, you can refer instance methods also.*

1. *Reference to a constructor*

*You can refer a constructor by using the new keyword.*

1. ***Functional Interface***

*An interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain one only abstract method. Functional interface is also known as single abstract method interface or SAM interface. Java provides predefined functional interfaces to deal with functionality programming by using lambda and method reference.*

1. ***Stream API***

*Java provides a new additional package in Java 8 called java.util.stream. This package consist of classes, interface, enum to allows functional-style operations on the elements. You can stream by importing java.util.stream package. Stream provide following features:*

1. *Stream doesn’t store elements.*
2. *Stream is functional in nature.*
3. *Stream is lazy and evaluates code only when required.*
4. *The elements of stream are only visited once during the life of stream.*

*Java stream provides a method filter () to filter stream element of the basis of given predicate. The signature of stream filter () method is given below:*

***Stream<T> filter (Predicate <? Super T> predicate)***

***Predicate :*** *it takes predicate reference as an argument. Predicate is functional interface. So you can pass lambda expression here. It returns a new stream.*

1. ***Base64 Encode Decode***

*Java provides a class Base64 to deal with encryption. You can encrypt and decrypt your data by using provided methods.*

1. *Basic / URL/ Filename encoding and decoding*

*It uses the Base64 alphabet specified by Java in RFC 4648 for encoding and decoding operations. Encoder doesn’t add any line separator character. The decoder rejects data that contains characters outside the base64 alphabet.*

1. *MIME*

*The encoded output must be represented in lines of no more than 76 characters each and uses a carriage return ‘/r’ followed immediately by linefeed ‘/n’ as the line separator.*

1. ***Java default methods***

*Java provides a facility to create default methods inside the interface. Methods which are defined inside the interface and tagged with default are known as default methods.*

1. ***Static method in interface***

*You can also define static methods inside the interface.*

1. ***Optional class***

*Java introduced a new class optional in jdk8. It is a public final class and used to deal with NullPointerException in Java application. You must import java.util package to use this class. It provides methods which are used to check the presence of value for particular variable.*

1. ***Collectors class***

*Collectors is a final class that extends Object class. It provides reduction operations, such as accumulating elements into collection, summarizing element according to various criteria, etc.*

1. ***For Each () method***

*Java provides a new method forEach () to iterate the elements. It is defined in iterable and stream interface. This method takes a single parameter which is a functional interface. So, you can pass lambda expression as an argument.*

*Along with foreach () method, java provides one more method forEachOrdered (). It is used to iterate elements in the order specified by the stream.*

1. ***String Joiner***

*Java added a new final class StringJoiner in java.util package. It is used to construct a sequence of characters separated by delimiter. Now, you can create string by passing delimiters like comma (,), hyphen (-) etc. You can also pass prefix and suffix to the char sequence.*

1. ***Nashorn Javascript engine***

*Nashorn is a Java Script engine. It is used to execute JavaScript code dynamically at JVM (Java Virtual Machine). Java provides a command-line tool jjs which is used to execute JavaScript code.*

1. ***Parallel Sorting Array***

*Java provides a new additional feature in array class which is used to to sort array elements parallel. New method has added to java.util.Array package that use the JSR 166 Fork/Join parallelism common pool to provide sorting arrays in parallel.*

1. ***Java Type Inference***

*Type inference is a feature of Java which provides ability to compiler to look at each method invocation and corresponding declaration determine the type of arguments.*

1. ***Method Parameter Reflection***

*Java provides new feature in which you can get the names of formal parameters of any method or constructor. The java.lang.reflect package contains all the required classes like method and parameter to work with parameter reflection.*

1. *Method class*

*It provides information about single method on a class or interface. The reflected method may be a class method or an instance method.*

1. *Parameter class*

*Parameter class provides information about method parameters, including its name and modifiers. It also provides an alternate means of obtaining attributes for the parameter.*

1. ***Java Type and Repeating Annotations***
2. *Java type annotations*

*Java 8 has included two new features repeating and type annotations in its prior annotations topic. After releasing java 8 annotations can be applied to any type use. It means that annotations can be used everywhere you use a type.*

1. *Java repeating annotations*

*Java allows you to repeating annotations in your source code. It is helpful when you want to reuse annotation for the same class. You can repeat an annotation anywhere that you would use a standard annotation. In order for the compiler to do this, two declarations are required in your code:*

1. *Declare a repeatable annotation type*

*Declaring of repeatable annotation type must be marked with the @Repeatable meta-annotation.*

1. *Declare the containing annotation type*

*Containing annotation type must have a value element with an array type. The component type of the array type must be the repeatable annotation type.*

1. ***JDBC Improvements***

*In java 8, java made major two major changes in JDBC API.*

1. *The JDBC-ODBC Bridge has been removed*

*Oracle doesn’t support the JDBC-ODBC Bridge. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.*

1. *Added some new features in JDBC 4.2*
2. *Addition of REF\_CURSOR support*
3. *Addition of java.sql.DriverAction interface*

*It is an interface that must be implemented when a Driver wants to be notified by DriverManager.*

1. *Addition of security check on deregisterDriver method in DriverManager class*

*The deregister method is intended only to be used by JDBC Drivers and not by applications.*

1. *Addition of the java.sql.SQLType interface*

*The interface is used to identify a generic SQL type, JDBC type or a vendor specific data type.*

1. *Addition of the java.sql.JDBCType enum*

*It is enumeration which defines the constants that are used to identify generic SQL types, called JDBC types. It extends java.lang.Enum and implements java.sql.SQLType.*

1. *Add support for large update counts*
2. *Changes to the existing interface*
3. *Rowset 1.2: Lists the enhancements for JDBC RowSet.*
4. ***Java Advance***
5. ***Data Structure***

*The data structures provided by the Java utility package are very powerful and perform a wide range of functions. These data structures consist of the following interface and classes:*

1. *Enumeration*

*The enumeration interface defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects.*

1. *BitSet*

*The BitSet class creates a special type of array that holds bit value. The BitSet array can increase in size as needed.*

1. *Vector*

*Vector implements a dynamic array. It similar to ArrayList, but with two differences*

1. *Vector is synchronized*
2. *Vector contains many legacy methods that are not part of the collections framework.*
3. *Stack*

*Stack is subclass of vector that implements a standard last-in-first-out (LIFO) stack. Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by vector, and adds several of its own.*

1. *Dictionary*

*Dictionary is an abstract class that represents a key/value storage repository and operates much like Map. The dictionary class is absolute. You should implement the Map interface to obtain key/value storage functionality.*

1. *Hashtable*

*Hashtable was part of the original java.util and is a concrete implementation of a Dictionary. Like HashMap, Hashtable stores key/value pairs in hash table.*

1. *Properties*

*Properties is subclass of Hashtable. It is used to maintain lists of values in which the key is a String and the value is also a String.*

1. ***Collections***

*The collections framework was designed to meet several goals, such as:*

1. *The framework had to be high-performance.*
2. *The framework had to allow different types of collections to work in a similar manner and with a high degree of interoperability.*
3. *The framework had to extend and/or adapt a collection easily.*

*A collection framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:*

1. *Interface*

*These are abstract data types that represent collections.*

1. *Implementation, i.e., classes*

*These are concrete implementations of collection interface.*

1. *Algorithms*

*These are the methods that perform useful computations, such as searching and sorting, on object that implements collections interface.*

1. ***Generics***

*Java generics methods and generic classes enable programmer to specify, with a single method declaration, a set of related methods, or with a single class declarations, a set of related types, respectively.*

1. *Generic methods*

*You can write a single generic method declaration that can be called with arguments of different types. The rules to define generic methods:*

1. *All generic method declarations have a type parameter section delimited by angle bracket (< and >) that precedes the methods return type (< E > in the next example).*
2. *Each type parameter section contains one or more type parameters separated commas.*
3. *The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to generic method, which are known as actual type arguments.*
4. *Bounded type parameters*

*There may be times when you’ll want to restrict the kinds of types that are allowed to be passed to a type parameter. For example, a method that operates on numbers might only want to accept instances of number or its subclasses. This is what bounded type parameters are for.*

*To declare a bounded type parameter, list the type parameter’s name, followed by the extends keyword, followed by its upper bond.*

1. *Generic classes*

*A generic class declarations look like a non-generic class declaration, except that the class name is followed by a type parameter section.*

*As with generic methods, the type parameters section of generic class can have one or more type parameters separated by commas. These classes are known as parameterized classes or parameterized types because they accept one or more parameters.*

1. ***Serialization***

*Java provides mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the objects data as well as information about the objects type and the type of data stored in the object.*

*After a serialized object has been written into a file, it can be read from the file and deserialized that it, the type information and bytes that represent the object and its data can be used to recreate the object in the memory.*

*Most impressive is that the entire process is JVM independent, meaning an object can be serialized on one platform and deserialized on an entirely different platform.*

1. ***Networking***

*The term network programming refers to writing programs that execute across multiple devices (computer), in which the devices are all connected to each other using a network.*

*The java.net package provides support for two common network protocols:*

1. *TCP*

*TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which referred to as TCP/IP.*

1. *UDP*

*UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.*

*This chapter gives a good understanding on the following two subjects:*

1. *Socket programming*

*Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.*

*When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket. The following steps occur when establishing a TCP connection between two computers using sockets:*

1. *The server instantiates a ServerSocket object, denoting which port numbers communication is to occur on.*
2. *The server invokes the accept() method of the ServerSocket class. This method waits until a client connect to the server on the given port.*
3. *After the server is waiting, a client instantiates a Socket object , specifying the server name and the port number to connect to.*
4. *The constructor of the Socket class attempt to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.*
5. *On the server side, the accept() method returns a reference to a new socket on the server that is connected to the clients socket.*
6. *ServerSocket Class Methods*
7. *Socket Class Methods*
8. *InetAddress Class Methods*
9. *Socket Client Example*
10. *Socket Server Example*
11. *URL Processing*

*URL stands for Uniform Resource Locator and represents a resource on the World Wide Web, such as a Web page or FTP directory.*

1. *Constructors*

*The java.net.URL class represents a URL and has a complete set of methods to manipulate URL in java.*

1. *URLConnections Class Methods*

*The openConnection() method returns a java.net.URLConnection, an abstract class whose subclasses represent the various types of URL connections. For example:*

1. *If you connect to a URL whose protocol is HTTP, the openConnection() method returns an HttpURLConnection object.*
2. *If you connect to a URL that represent a JAR file, the openConnection() return a JarURLConnection object, etc.*
3. ***Sending Email***

*To send an email using your Java Applications is simple enough but to start with you have JavaMail API and Java Activation Framework (JAF) installed on your machine.*

1. ***Multithreading***

*Java is a multithreaded programming language, which means we can develop multi threaded programs using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.*

1. ***Applet Basic***

*An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. Life cycle of an applet:*

1. *Init*

*This method is intended for whatever initialization is needed for your applet.*

1. *Start*

*This method is automatically called after the browser calls the init method.*

1. *Stop*

*This method is automatically called when the user moves off the page on which the applet sits.*

1. *Destroy*

*This method is only called when the browser shut downs normally.*

1. *Paint*

*Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser.*

1. ***Documentation***

*The java language supports three types of comment:*

1. */\* text \*/*

*The compiler ignores everything from /\* to \*/.*

1. *//text*

*The compiler ignores everything from // to end of the line.*

1. */\*\* documentation \*\*/*

*This is a documentation comment and in general is called doc comment.*

1. ***Spring Framework***

*Spring is a lightweight, loosely coupled, and integrated framework for developing enterprise applications in java.*

1. ***Advantage***

*The advantages of spring framework:*

1. *Predefined template*

*Spring framework provides templates for JDBC, Hibernate, JPA, etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.*

1. *Loose coupling*

*The spring application are loosely coupled because of dependency injection.*

1. *Easy to test*

*The dependency injection makes easier to test the application. The EJB and Struts application require server to run the application but Spring framework doesn’t require server.*

1. *Lightweight*

*Spring framework is lightweight because of its POJO implementation. The Spring framework doesn’t force the programmer to inherit any class or implements any interface. That is why it is said non-invasive.*

1. *Fast development*

*The dependency injection feature of Spring framework and it support to various frameworks makes the easy development of JavaEE application.*

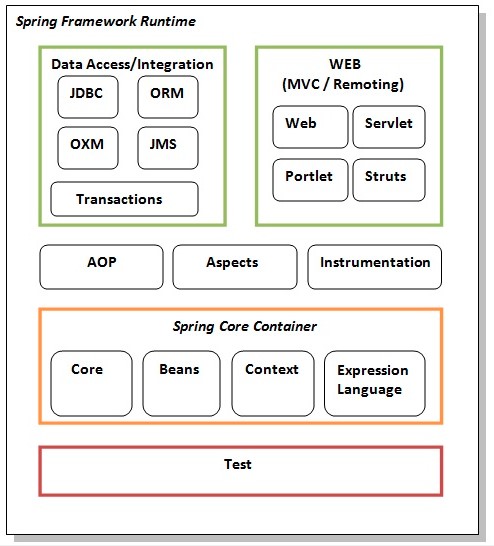
1. *Powerful abstraction*

*It provides powerful abstraction to Java EE specifications such as JMS, JDBC, JPA, JTA.*

1. *Declarative support*

*It provides declarative support for caching, validation, transaction and formatting.*

1. ***Module***

**

1. *Test*

*This layer provides support of testing with JUnit and TestNG.*

1. *Spring Core Container*

*The spring core container contains core, beans, context, and expression language (EL) modules.*

1. *Core and beans*

*These modules provide IOC and dependency injection features.*

1. *Context*

*This module supports internationalization (i18N), EJB, JMS, Basic Remoting.*

1. *Expression Language*

*It is an extension to the EL defined in JSP. It provides support to setting and getting property values, method invocation, accessing collections, and indexers, named variables, logical, arithmetic, operators, retrieval of object by name etc.*

1. *AOP, Aspects and Instrumentation*

*These modules support aspect oriented programming implementations where you can use advices, point cuts etc. To decouple the code.*

*The aspects module provides support to integration with AspectJ.*

*The instrumentation module provides support to class instrumentation and classloader implementations.*

1. *Data Access/ Integration*

*This group comprises of JDBC, ORM, OXM, JMS and transaction modules. These modules basically provide support to interact with the database.*

1. *Web*

*This group comprises of Web, Web-Servlet, Web-Struts, and Web-Portlet. These modules provide support to create web application.*

1. ***IOC ( Inversion of Control) and DI (Dependency Injection)***

*Is a design pattern to provide loose coupling, it removes the dependency from the program. The IOC container is responsible to instantiate, configure, and assemble the object.*

1. *IOC (Inversion of Control)*

*Role of IOC container in spring:*

1. *Create an instance*
2. *Configure the instance/object*
3. *Assemble the dependencies between the object*

*Types of IOC container in spring:*

1. *Bean Factory*

*The XmlBeanFactory is the implementation class for the BeanFactory interface. To use BeanFactory we need create the instance of XmlBeanFactory class.*

1. *Application Context*

*The application context interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Springs AOP, message resource handling (I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application.*

*BeanFactory is the basic container whereas ApplicationContext is the advanced container. ApplicationContext extends the BeanFactory interface. ApplicationContext provides more facilities than BeanFactory such as integration with spring AOP, message resource handling for I18n etc.*

1. *DI (Dependency Injection)*
2. *Dependency Lookup*

*The dependency lookup is an approach where we get the resource after demand. There can be various ways to get the resource for example. Problems:*

1. *Tight coupling*

*The dependency lookup approach makes the code tightly coupled. If resource is changed, we need to perform a lot of modification in the code.*

1. *Not easy for testing*

*This approach creates a lot of problems while testing the application especially in black box testing.*

1. *Dependency Injection*

*The Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. In such case we provide the information from the external source such as XML file. It make or code loosely coupled and easier for testing*

1. *Two Ways to Perform Dependency Injection in Spring Framework*
2. *By Constructor*
3. *By Method*
4. ***Difference between constructor and setter injection***

*There are many key differences between constructor injection and setter injection:*

1. *Partial dependency*

*Can be injected using setter injection but it is not possible by constructor.*

1. *Overriding*

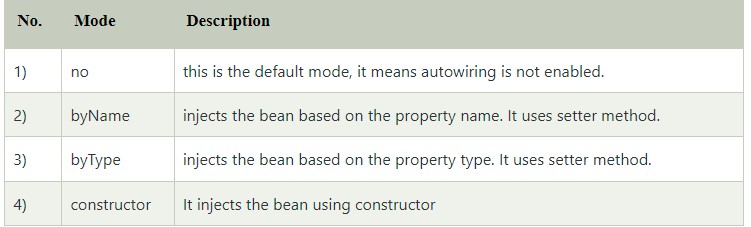
*Setter injection overrides the constructor injection. If we use both constructor and setter injection, IOC container will use the setter injection.*

1. *Changes*

*We can easily change the value by setter injection. It doesn’t create a new bean instance always like constructor. So setter injection is flexible than constructor injection.*

1. ***Autowiring***

*Autowiring enables the programmer to inject the bean automatically. We don’t need to write explicit injection logic, it requires the less code. It can’t be used for primitive and String values. Disadvantage of autowiring are no control of programmer. The autowiring modes are given below:*

**

1. ***Bean Scope***

*There are 5 bean scopes in spring framework:*

1. *Singleton*

*The bean instance will be only once and same instance will be returned by the IOC container. It is default scope.*

1. *Prototype*

*The bean instance will be created each time when requested.*

1. *Request*

*The bean instance will be created per HTTP request.*

1. *Session*

*The bean instance will be created per HTTP session.*

1. *Global Session*

*The bean instance will be created per HTTP global session. It can be used in portlet context only.*

*Singleton scope should be used with EJB stateless session bean and prototype scope with EJB stateful session bean.*

1. ***Transaction Management***

*Spring framework provides two types of transaction management supports:*

1. *Programmatic Transaction Management*

*Should be used for few transaction operations.*

1. *Declarative Transaction Management*

*Should be used for many transaction operations.*

1. ***Spring JDBC***
2. *Advantages*

*Less code, by using the JdbcTemplate class, you don’t need to create connection, statement, start transaction, commit transaction and close connection to execute different queries. You can execute query directly.*

1. *Classes*

*There are classes for spring JDBC:*

1. *JdbcTemplate*

*It is the central class in the Spring JDBC support classes. It takes care of creation and release of resources such as creating and closing of connection object etc.*

*You can fetch records from the database by the query method of JdbcTemplate. There are two interface to do this:*

1. *ResultSetExtractor*
2. *RowMapper*
3. *SimpleJdbcTemplate*

*Spring 3 JDBC supports the Java 5 feature var-args (variable argument) and autoboxing by the help of SimpleJdbcTemplate.*

1. *NamedParameterJdbcTemplate*

*NamedParameterJdbcTemplate class is used to pass value to the named parameter. A named parameter is better than*

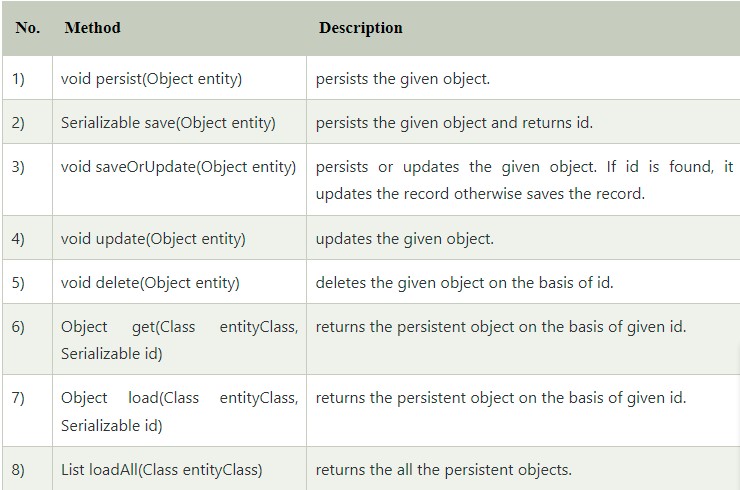
1. *SimpleJdbcInsert*
2. *SimpleJdbcCall*
3. *ORM framework*

*Spring provides API to easily integrate spring with ORM framework such as Hibernate, JPA (Java Persistence API), JDO (Java Data Objects), Oracle Toplink and iBATIS.*

1. *Advantage*
2. *Less coding is required*
3. *Easy to test*
4. *Better exception handling*
5. *Integrated transaction management*
6. *Hibernate*

*We can simply integrate hibernate application with spring application. In hibernate framework, we provide all the database information hibernate.cfg.xml file. But if we are going to integrate the hibernate application with spring, we don’t need to create the hibernate.cfg.xml file. We can provide all the information in the applicationContext.xml file.*

*The Spring framework provides HibernateTemplate class, so you don’t need to follow so many steps like create Configuration, BuildSessionFactory, Session, beginning, and committing transaction etc. Methods of HibernateTemplate class:*

**

1. *JPA*

*Spring data JPA API provides JpaTemplate class to integrate spring application with JPA. JPA (Java Persistent API) is the sun specification for persisting objects in the enterprise application. It is currently used as the replacement for complex entity beans. The implementation of JPA specification are provided by many vendors such as hibernate, toplink, iBatis, OpenJPA etc.*

*Advantage of Spring JpaTemplate is you don’t nedd to write the before and after code for persisting, updating, deleting or search object such as creating Persistence instance, creating EntityManagerFactory instance, creating EntityTransaction instance, creating EntityManager instance, committing EntityTransaction instance and closing EntityManager.*

1. ***Spring AOP***

*Aspect Oriented Programming (AOP) compliments OOPs in the sense that it also provides modularity. It is a methodology that divides the program logic into pieces or parts or concern. But the key unit of modularity is aspect than class. AOP breaks the program logic into distinct parts (called concern). It is used to increase modularity by cross-cutting concerns. A cross-cutting concern is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.*

*AOP concepts and terminology:*

1. *Join Point*

*Join point is any point in your program such as method execution, exception handling, field access etc. Spring supports only method execution joint point.*

1. *Advice*

*Advice represents an action taken by an aspect at a particular join point. There are different types advices:*

1. *Before advice*

*It executes before a join point*

1. *After returning advice*

*It execute after a join point completes normally.*

1. *After throwing advice*

*It executes if method exist by throwing an exception.*

1. *After finally advice*

*It executes after a join point regardless of join point exit whether normally or exceptional return.*

1. *Around advice*

*It executes before and after a joint point.*

1. *Pointcut*

*It is an expression language of AOP that matches join points.*

1. *Introduction*

*It means introductional of additional method and fields for a type. It allows you to introduce new interface to any advised object.*

1. *Target Object*

*It is the object i.e. being advised by one or more aspects. It is also known as proxied object in spring because Spring AOP implemented using runtime proxies.*

1. *Aspect*

*It is class that contains advices, joinpoints etc.*

1. *Interceptor*

*It is an aspect that contains only one advice.*

1. *AOP Proxy*

*It is used to impelement aspect contracts, created by AOP framework.*

1. *Weaving*

*It is the process of linking aspect with other application types or objects to create an advised object. Weaving can be done at compile time, load time or runtime. Spring AOP weaving at runtime.*

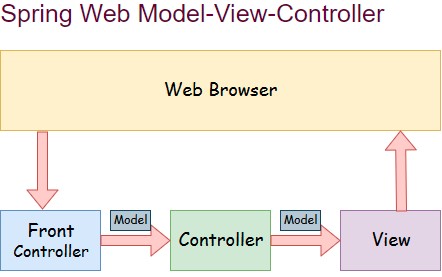
*AOP implementations:*

1. *AspectJ*
2. *Spring AOP*
3. *JBOSS AOP*

*The Advantages are AOP enables you to dynamically add or remove concern before or after the business logic. It is pluggable and easy to maintain.*

1. ***Spring MVC***

*Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.*

**

1. *Model*

*A model contains the data of application. A data can be a single object or collection of a objects.*

1. *Controller*

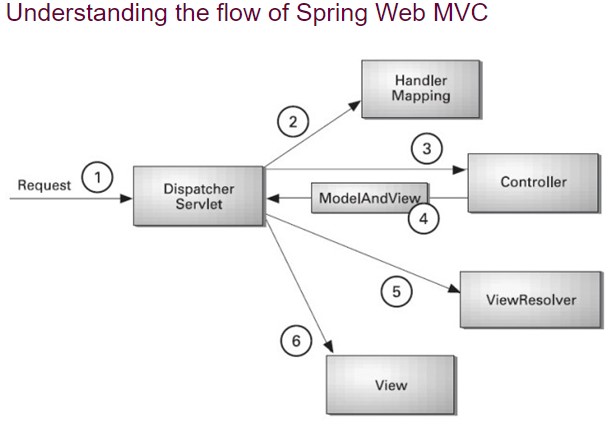
*A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.*

1. *View*

*A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page.*

1. *Front Controller*

*In spring web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.*

**

*Advantages of Spring MVC framework:*

1. *Separate roles*
2. *Light weight*
3. *Powerful configuration*
4. *Rapid development*
5. *Reusable business code*
6. *Easy to test*
7. *Flexible mapping*
8. ***Spring Annotation***
9. ***Core Spring Framework Annotations***
10. ***@Required***

*This annotation is applied on bean setter method. Consider a scenario where you need to enforce a required property.*

1. ***@Autowired***

*This annotation is applied on fields, setter method, and constructor. The @autowired annotation injects object dependency implicity. When you use @autowired on fields and pass the values for the fields using the property name, spring will automatically assign the fields with the passed values.*

1. ***@Qualifier***

*This annotation is used along with @Autowired annotation. When you need more control of the dependency injection process, @Qualifier can be used. @Qualifier can be specified on individual constructor arguments or method parameters. This annotation is used to avoid confusion which occurs when you create more than one bean of the same type and want to wire only one of them with a property.*

1. ***@Configuration***

*This annotation is used on classes which define beans. @Configurations is an analog for XML configuration file, it is configuration using Java class. Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies.*

1. ***@ComponentScan***

*This annotation is used with @Configuration annotation to allow spring to know the packages to scan for annotated components. @ComponentScan is also used to specify base packages using basePackageClasses or basePackage attributes to scan. If specific packages are not defined, scanning will occur from the package of the class that declares this annotation.*

1. ***@Bean***

*This annotations is used at the method level. Spring calls these methods when a new instance of the return type is required. The resulting bean has the same name as the factory method. @Bean annotations works with @Configuration to create spring beans. As mentioned early, @Configuration will have methods to instantiate and configure dependencies.*

1. ***@Lazy***

*This annotation is used on component classes. By default all autowired dependencies created and configure at start up. But if you want to initialize a bean lazily, you can use @Lazy annotation over the class. This means that the bean will be created and initialized only when it is first requested for. You can also use this annotation on @Configuration class. This indicates that all @Bean methods within that @Configuration should be lazily initialized.*

1. ***@Value***

*This annotation is used at the field, constructor parameter, and method parameter level. The @Value annotation indicates a default value expression for the field or parameter to initialize the property with. As the @Autowired annotation tells spring to inject object into another when it loads your application context, you can also use @Value annotation to inject values from property file into a bean’s attribute. It support both #{…} and ${…} placeholders.*

1. ***@DependsOn***

*We can use this annotation to make Spring* ***initialize other beans before the annotated one.*** *Usually, this behaviour is automatic, based on the explicit dependencies between beans.*

*We only need this annotation when the dependencies are implicit, for example, JDBC driver loading or static variable initialization.*

*We can use @DependsOn the dependent class specifying the names of the dependency beans. The annotations value arguments needs an array containing the dependency bean names.*

1. ***@LookUp***

*A method annotated with @LookUp tells Spring to return an instance of the methods return type when we invoke it.*

1. ***@Primary***

*Sometimes we need to define multiple beans of the same type. In these cases, the injection will be unsuccessful because spring has no clue which bean we need.*

*We already saw an option to deal with the scenario: marking all the wiring points with @Qualifier and specify the name of the required bean.*

*However, most of the time we need a specific bean and rarely the others. We can use @Primary to simplify this case: if we mark the most frequently used bean with @Primary it will be chosen on unqualified injection points.*

1. ***@Scope***

*We use @Scope to define the scope of a @Component class or a @Bean definition. It can be either singleton, prototype, request, session, globalSession or some custom scope.*

1. *Context Configuration Annotations*
2. ***@Profile***

*If we want Spring to use a @Component class or a @Bean method only when a specific profile is active, we can mark it with @Profile. We can configure the name of the profile with the value argument of the annotation.*

1. ***@Import***

*We can use specific @Configuration classes without component scanning with this annotation. We can provide those classes with @Import value argument.*

1. ***@ImportResource***

*We can import XML configurations with this annotation. We can specify the XML file locations with the locations argument, or with its alias, the value argument.*

1. ***@PropertySource, @PropertySources***

*With this annotation, we can define property files for application settings. @PropertySource leverages the Java 8 repeating annotations feature, which means we can mark a class with it multiple times.*

*We can use this annotation to specify multiple @PropertySource configurations.*

1. ***Spring Framework Stereotype Annotations***
2. ***@Component***

*This annotation is used on classes to indicate a spring component. The @Component annotation marks the Java class as a bean or say component so that component-scanning mechanism of spring can add into the application context.*

1. ***@Controller***

*The @Controller annotation is used to indicate the class is a spring controller. This annotation can be used to identify controllers for spring MVC or spring WebFlux.*

1. ***@Service***

*This annotation is used on a class. The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs. This annotation is a specialized form of the @Component annotation intended to be used in the service layer.*

1. ***@Repository***

*This annotation is used on Java classes which directly access the database. The @Repository annotation works as marker for any class that fulfills the role of repository or Data Access Object.*

1. ***Spring Web***
2. ***@RequestMapping***

*@RequestMapping marks request handler methods inside @Controller classes, it can be configured using:*

1. *Path*

*Or it aliases, name and value, which URL the method is mapped to.*

1. *Method*

*Compatible HTTP method.*

1. *Params*

*Filters request based on presence, absence, or value of HTTP parameters.*

1. *Headers*

*Filters request based on presence, absence, or value of HTTP headers.*

1. *Consumes*

*Which media types the method can consume in the HTTP request body.*

1. *Produces*

*Which media types the method can produce in the HTTP response body.*

*Spring 4.3 introduced five new and more specific annotations for each HTTP request type:*

1. ***@GetMapping***

*The @GetMapping annotation is specialized version of @RequestMapping annotation that acts as a shortcut for (method = RequestMethod.GET).*

1. ***@PostMapping***

*The @PostMapping annotation is specialized version of @RequestMapping annotation that acts as a shortcut for (method = RequestMethod.POST).*

1. ***@PutMapping***

*The @PutMapping annotation is specialized version of @RequestMapping annotation that acts as a shortcut for (method = RequestMethod.PUT).*

1. ***@PatchMapping***

*The @PatchMapping annotation is specialized version of @RequestMapping annotation that acts as a shortcut for (method = RequestMethod.PATCH).*

1. ***@DeleteMapping***

*The @DeleteMapping annotation is specialized version of @RequestMapping annotation that acts as a shortcut for (method = RequestMethod.DELETE).*

1. ***@RequestBody***

*Which maps the body of the HTTP request to an object. The deserialization is automatic and depends on the content type of the request.*

1. ***@PathVariable***

*This annotations indicates that a method argument is bound to a URI template variable. We can specify the URI template with the @RequestMapping annotation and bind a method argument to one of the template part with @PathVariable. Moreover, we can mark a path variable optional by setting the argument required to false.*

1. ***@RequestParam***

*We use @RequestParam for accessing HTTP request parameters. It has the same configuiration option as the @PathVariable annotation. In addition to those setting, with @RequestParam we can specify an injected value when Spring finds no or empty value in the request. To achieve this, we have to set defaultValue argument.*

*Besides parameters, there are* ***other HTTP request parts we can access: cookies and headers****. We can access them with the annotations* ***@CookieValue and @RequestHeader*** *respectively. We can configure them the same way as @RequestParam.*

1. ***@ResponseBody***

*If we mark a request handler method with @ResponseBody,* ***Spring treats the result of the method as the response itself.***

*If we annotate a @Controller class with this annotation, all request handler methods will use it.*

1. ***@ExceptionHandler***

*With this annotation, we can declare a custom error handler method. Spring calls this method when a request handler method throws any of specified exceptions. The caught exception can be passed to the methods as an arguments.*

1. ***@ResponseStatus***

*We can specify the desired HTTP status of the response if we annotate a request handler method with this annotation. We can declare the status code with the code argument. Or its alias, the value argument.*

*Also, we can provide a reason using the reason argument. We also use it along with @ExceptionHandler.*

1. ***@RestController, @RestControllerAdvice***

*The @****RestController*** *combines* ***@Controller and @ResponseBody.***

*The @****RestControllerAdvice*** *applied on Java classes. This is a convenience annotation which combines @ControllerAdvice and @ResponseBody, used along @ExceptionHandler annotation to handle exception that occur within the controller*

1. ***@ModelAttribute***

*@ModelAttribute is an annotation that binds a method parameter or method return value to a named model attribute, and then exposes it to a web view. With this annotation we can access elements that are already in the model of an MVC @Controller, by providing model key.*

*Like with @PathVariable and @RequestParam, we don’t have to specify the model key if the argument has the same name.*

1. ***@CrossOrigin***

*@CrossOrigin enables cross-domain communication for the annotated request handler methods. If we mark a class with it, it applies to all request handler methods in it. We can fine-tune CORS behaviour with this annotations arguments.*

1. ***@MappingAttributes***
2. ***@InitBinder***

*This annotation is a method level annotation that plays the role of identifying the methods which initialize the WebDataBinder – a DataBinder that binds the request parameter to JavaBean objects. To customize request parameter data binding, you can use @InitBinder annotated methods within our controller. The methods annotated with this all argument types that handler methods support.*

1. ***@Mappings and @Mapping***

*This annotation is used on fields. The @Mapping annotation is a meta annotation that indicates a web mapping annotation. When mapping different field names, you need to configure the source field to its target field to its target field and to do that you have to add the @Mapping annotation.*

*@****Mappings*** *accept an array of @Mapping having the source and the target field.*

1. ***@MatrixVariable***

*This annotation is used to annotate request handler method arguments so that Spring can inject the relevant bits of matrix URI. Matrix variables can appear on any segment each separated by a semicolon.*

1. ***@RequestAttribute***

*This annotation is used to bind the request attribute to handler method parameter. Spring retrieves the named attributes value to populate the parameter annotated with @RequestAttribute. While the @RequestParam annotation is used bind the parameter values from query string , the @RequestAttribute is used to access the objects which have been populated on the server side.*

1. ***@ControllerAdvice***

*A controller advice allows you to use exactly the sampe exception handling techniques but apply them across the whole application, not just to an individual controller.*

1. ***@SessionAttribute, @SessionAttributes***

*The* ***@SessionAttribute*** *is used at method parameter level. The @SessionAttribute used to bind the method parameter to a session attribute. This annotation provides a convenient access to the existing or permanent session attributes.*

*The @****SessionAttributes*** *is applied at type level for a specific handler, used when you want add a JavaBean object into a session. This is used when you want to keep the object in session for short lived*

1. ***Spring Cloud Annotation***
2. ***@EnableConfigServer***

*The* ***@EnableConfigServer*** *is used at the class level, this annotation to start a config server that the other applications can talk to.*

1. ***@EnableEurekaServer***

*The* ***@EnableEurekaServer*** *is used at the class level, this annotation represent Netflix Eureka implementation of a discovery server and integration is provided by Spring Boot.*

1. ***@EnableDiscoveryClient***

*The* ***@EnableDiscoveryClient*** *is used at the class level, used to tell any application to register itself with Eureka you just need to add this annotation to the application entry point.*

1. ***@EnableCircuitBreaker***

*The* ***@EnableCircuitBreaker*** *is applied on Java classes that can act as the circuit breaker. The circuit breaker pattern can allow a micro service continue working when a related service fails, preventing the failure from cascading. The class annotated with this will monitor, open, and close the circuit breaker.*

1. ***@HystrixCommand***

*The* ***@HystrixCommand*** *is used at the method level. Hystrix looks for any method annotated with this annotation and wraps it into a proxy connected to a circuit breaker so that Hystrix can monitor it.*

1. ***Spring Data JPA Annotations***
2. ***@Access***

*@****Access*** *annotation is used to specify the access type of the associated entity class, mapped supperclass, or the embeddable class and entity attribute.*

1. ***@AssociationOverride, @AssociationOverrides***

*The @****AssociationOverride*** *annotation is used to override an association mapping (e.g @ManyToOne, @OneToOne, @OneToMany, @ManyToMany) inherited from a mapped superclass or an embeddable.*

*The @****AssociationOverrides*** *is used to group several @AssociationOverride annotations.*

1. ***@AttributeOverride, @AttributeOverrides***

*The @****AttributeOverride*** *annotation is used to override an attribute mapping inherited from a mapped superclass or an embeddable.*

*The @****AttributeOverrides*** *is used to group several @AttributeOverride annotations.*

1. ***@Basic***

*The @****Basic*** *annotation is used to map a basic attribute type to a database column.*

1. ***@Cacheable***

*The @****Cacheable*** *annotation is used to specify whether an entity should be stored in the second level cache.*

1. ***@CollectionTable***

*The @****CollectionTable*** *annotation is used to specify the database table that stores the values of a basic or an embeddable type collection.*

1. ***@Column, @ColumnResult, @ConstructorResult***

*The @****Column*** *annotation is used to specify the mapping between a basic entity attribute and the database table column.*

*The @****ColumnResult*** *is used in conjunction with @SqlResultMapping or @ConstructorResult annotations to map a SQL column for a given SELECT query.*

*The @****ConstructorResult*** *is used in conjuction with the @SqlResultMapping annotations to map columns of a given SELECT query to a certain object constructor.*

1. ***@Convert, @Converter, @Converts***

*The @****Convert*** *annotation is used to specify the AttributeConverter implementation used to convert the currently annotated basic attribute.*

*The @****Converter*** *is used to specify that the current annotate AttributeConverter implementation can be used as a JPA basic attribute converter.*

*The @****Converts*** *is used to group multiple @Convert annotations.*

1. ***@CreatedBy, @LastModifiedBy, @CreatedDate, @LastModifiedDate***

*The* ***@CreatedBy*** *is used to automate user create on field.*

*The* ***@LastModifiedBy*** *is used to automate last modified user on field.*

*The* ***@CreatedDate*** *is used to automate create date on field.*

*The* ***@LastModifiedDate*** *is used to automate last modified date on field.*

1. ***@DiscriminatorColumn, @DiscriminatorValue***

*The @****DiscriminatorColumn*** *is used to specify the discriminator column name and the discriminator type for the SINGLE\_TABLE and JOINED inheritance strategies.*

*The @****DiscriminatorValue*** *is used to specify what value of the discriminator column is used for mapping the currently annotated entity.*

1. ***@ElementCollection***

*The @****ElementCollection*** *is used to specify collection of basic or embeddable types.*

1. ***@Embeddable, @Embedded, @EmbeddedId***

*The @****Embedabble*** *is used to specify embeddable types. Like basic types, embeddable types do not have any identity, being managed by their owning entity.*

*The @****Embedded*** *is used to specify that a given entity attribute represents an embeddable type.*

*The @****EmbeddedId*** *is used to specify the entity identifiers is an embeddable type.*

1. ***@EnableJpaRepositories***

*The @****EnableJpaRepository*** *is used to scan the package of the annotated configuration class for Spring Data repositories by default.*

1. ***@Entity, @EntityListeners, @EntityResult***

*The @****Entity*** *is used to specify that the currently annotate class represent an entity type. Unlike basic and embeddable types, entity types have an identity and their state is managed by the underlying Persistence Context.*

*The @****EntityListeners*** *is used to specify an array of callback listeners classes that are used by the currently annotated entity.*

*The @****EntityResult*** *is used with @SqlResultSetMapping annotation to map the selected columns to an entity.*

1. ***@Enumerated***

*The @****Enumerated*** *is used to specify that an entity attribute represent an enumerated type.*

1. ***@ExcludeDefaultListeners, @ExcludeSuperclassListeners***

*The @****ExcludeDefaultListeners*** *is used to specify that the currently annotated entity skips the invocation of any default listeners.*

*The @****ExcludeSuperclassListeners*** *is used to specify that the currently annotated entity skips the invocation of listeners declared by its superclass.*

1. ***@FieldResult***

*The @****FieldResult*** *is used with the @EntityResult annotation to map the selected columns to the fields of some specific entity.*

1. ***@ForeignKey***

*The @****ForeignKey*** *is used to specify the associated foreign key of @JoinColumn mapping. The @ForeignKey annotation is only used if the automated schema generation tool is enabled. In which case, it allows you to customize the underlying foreign key definition.*

1. ***@GeneratedValue***

*The @****GeneratedValue*** *is used to specifies that the entity identifier value is automatically generated using an identity column, a database sequence, or a table generator. Hibernate supports the @GeneratedValue mapping even for UUID identifiers.*

1. ***@Id, @IdClass***

*The @****Id*** *is used to specifies the entity identifier. An entity must always have an identifier attribute, which is used when loading the entity in a given Persistence Context.*

*The @****IdClass*** *is used if the current entity defined a composite indentifier. A separate class encapsulates all the identifier attributes, which are mirrored by the current entity mapping.*

1. ***@Index***

*The @****Index*** *is used by the automated schema generation tool to create a database index.*

1. ***@Inheritance***

*The @****Inheritance*** *is used to specify the inheritance strategy of a given entity class hierarchy.*

1. ***@JoinColumn, @JoinColumns, @JoinTable***

*The @****JoinColumn*** *is used to specify the FOREIGN KEY column used when joining an entity association or an embeddable collection*

*The @****JoinColumns*** *is used to group multiple @JoinColumn annotations, which are used when mapping entity association or an embeddable collection using a composite identifier.*

*The @JoinTable is used to specify the link table between two other database tables.*

1. ***@Lob***

*The @****Lob*** *is used to specify that the currently annotated entity attribute represents a large object type.*

1. ***@Lock***

*The @****Lock*** *is used to specify the LockModeType to be used when executing the query. It will be evaluated when using Query on a query method or if you derive the query from the method name.*

1. ***@ManyToMany, @ManyToOne, @OneToMany, @OneToOne***

*The @****ManyToMany*** *is used to specify a many-to-many database relationship.*

*The @****ManyToOne*** *is used to specify a many-to-one database relationship.*

*The @****OneToMany*** *is used to specify a one-to-many database relationship.*

*The @****OneToOne*** *is used to specify a one-to-one database relationship.*

1. ***@MapKey, @MapKeyClass, @MapKeyColumn, @MapKeyEnumerated, @MapKeyJoinColumn, @MapKeyJoinColumns, @MapKeyTemporal***

*The @****MapKey*** *is used to specify the key of a java.util.Map association for which the key type is either the primary key or an attribute of the entity that represent the value of a map.*

*The @****MapKeyClass*** *is used to specify the key of a java.util.Map association.*

*The @****MapKeyColumn*** *is used to specify the database column, which stores the key of a java.util.Map association for which the map key is a basic type.*

*The @****MapKeyEnumerated*** *is used to specify that the key of java.util.Map association is Java Enum.*

*The @****MapKeyJoinColumn*** *is used to specify that the key of java.uti.Map association is an entity association. The map key column is a FOREIGN KEY in a link table that also joins the Map owners table with the table where the MAP value resides.*

*The @****MapKeyJoinColumns*** *is used to group several @MapKeyJoinColumn mappings when the java.util.Map association key uses a composite identifier.*

*The @****MapKeyTemporal*** *is used to specify that the key of java.util.Map association is a @TemporalType (e.g. DATE, TIME, TIMESTAMP).*

1. ***@MappedSuperclass, @MapsId***

*The @****MappedSuperclass*** *is used to specify that the currently annotated type attributes are inherited by any subclass entity.*

*The @****MapsId*** *is used to specify that the identifier is mapped by the currently annotated @ManyToOne or @OneToOne association.*

1. ***@Modifying***

*The @****Modifying*** *is used to enhance the @Query annotations so that we can execute not only SELECT queries, but also INSERT, UPDATE, DELETE, and even DDL queries.*

1. ***@NamedAttributeNode***

*The @****NamedAttributeNode*** *is used to specify each individual attribute node that needs to be fetched by an Entity Graph.*

1. ***@NamedEntityGraph, @NamedEntityGraphs***

*The @****NamedEntityGraph*** *is used to specify an Entity Graph that can be used by an entity query to override the default fetch plan.*

*The @****NamedEntityGraphs*** *is used to group @NamedEntityGraph annotations.*

1. ***@NamedNativeQueries, @NamedNativeQuery, @NamedQueries, @NamedQuery***

*The @****NamedNativeQueries*** *is used to group multiple @NamedNativeQuery annotations.*

*The @****NamedNativeQuery*** *is used to specify a native SQL query that can be retrieved later by its name.*

*The @****NamedQueries*** *is used to group multiple @NamedQuery annotations.*

*The @****NamedQuery*** *is used to specify a JPQL query that can be retrieved later by its name.*

1. ***@NamedStoredProcedureQueries,@NamedStoredProcedureQuery, NamedSubgraph***

*The @****NamedStoredProcedureQueries*** *is used to group multiple @NamedStoredPorocedureQuery annotations.*

*The @****NamedStoredProcedureQuery*** *is used to specify a stored procedure query that can be retrieved later by its name.*

*The @****NamedSubgraph*** *is used to specify a subgraph in an Entity Graph.*

1. ***@NoRepositoryBean***

*The* ***@NoRepositoryBean*** *is used to avoid creating repositories proxies for interfaces that actually match the criteria of a repo interface but are not intended to be one.*

1. ***@OrderBy, @OrderColumn***

*The @****OrderBy*** *is used to specify the entity attributes used for sorting when fetching the currently annotated collection.*

*The @****OrderColumn*** *is used to specify that the current annotation collection order should be materialized in the database.*

1. ***@Param***

*The* ***@Param*** *is used to give a method parameter to a concrete name and bind the name in query, works with both @Query and @NamedQuery.*

1. ***@PersistenceContext,@PersistenceContexts,@PersistenceProperty,PersistenceUnit, @PersistenceUnits***

*The @****PersistenceContext*** *is used to specify the EntityManager that needs to be injected as a dependency.*

*The @****PersistenceContexts*** *is used to group multiple @PersistenceContext annotations.*

*The @****PersistenceProperty*** *is used by the @PersistenceContext annotation to declare JPA provider properties that are passed to the underlying container when the EntityManager instance is created.*

*The @****PersistenceUnit*** *is used to specify the EntityManagerFactory that needs to be injected as a dependency.*

*The @****PersistenceUnits*** *is used to group multiple @PersistenceUnit annotations.*

1. ***@PostLoad, @PostPersist, @PostRemove, @PostUpdate***

*The @****PostLoad*** *is used to specify a callback method that fires after an entity is loaded.*

*The @****PostPersist*** *is used to specify a callback method that fires after an entity is persisted.*

*The @****PostRemove*** *is used to specify a callback method that fires after an entity is removed.*

*The @****PostUpdate*** *is used to specify a callback method that fires after an entity is updated.*

1. ***@PrePersist, @PreRemove, @PreUpdate***

*The @****PrePersist*** *is used to specify a callback method that fires before an entity is persisted.*

*The @****PreRemove*** *is used to specify a callback method that fires before an entity is removed.*

*The @****PreUpdate*** *is used to specify a callback method that fires before an entity is updated.*

1. ***@PrimaryKeyJoinColumn, @PrimaryKeyJoinColumns***

*The @****PrimaryKeyJoinColumn*** *is used to specify that the primary key column of the currently annotated entity is also a foreign key to some other entity (e.g. a base class table in a JOINED inheritance strategy , the primary table in a secondary table mapping, or the parent table in a @OneToOne relationship).*

*The @****PrimaryKeyJoinColumns*** *is used to group multiple @PrimaryKeyJoinColumn annotations.*

1. ***@Procedure***

*The @****Procedure*** *is used to map a user defined repository method to database stored procedure.*

1. ***@QueryHint, @Query***

*The @****QueryHint*** *is used to specify a JPA provider hint used by a @NamedQuery or @NamedNativeQuery annotation.*

*The @****Query*** *is used to define a query that we can execute, with this we can provide a JPQL implementation for a repository method.*

1. ***@SecondaryTable, @SecondaryTables***

*The @****SecondaryTable*** *is used to specify a secondary table for the currently annotated entity.*

*The @****SecondaryTables*** *is used to group multiple @SecondaryTable annotations.*

1. ***@SequenceGenerator***

*The @****SequenceGenerator*** *is used to specify the database sequence used by the identifier generator of the currently annotated entity.*

1. ***@SqlResultSetMapping. @ SqlResultSetMappings***

*The @****SqlResultSetMapping*** *is used to specify the ResultSet mapping of a native SQL query or stored procedure.*

*The @****SqlResultSetMapping****s is used to group multiple @SqlResultSetMapping annotation.*

1. ***@StoredProcedureParameter***

*The @****StoredProcedureParameter*** *is used to specify a parameter of a @NamedStoreProcedureQuery.*

1. ***@Table, @TableGenerator***

*The @****Table*** *is used to specify the primary table of the currently annotated entity.*

*The @****TableGenerator*** *is used to specify the database table used by the identity generator of the currently annotated entity.*

1. ***@Temporal***

*The @****Temporal*** *is used to specify the TemporalType of the currently annotated java.util.Date or java.util.Calendar entity attribute.*

1. ***@Transactional***

*The* ***@Transactional*** *is simply a metadata that can be consumed by some runtime infrastructure, placed before an interface definition, a method on an interface, a class definition, or a public method on a class.*

*The annotation further supports configuration like:*

1. *The propagation type of the transaction*
2. *The isolation level of the transaction*
3. *A timeout for the operation wrapped by the transaction*
4. *A read only flag – a hint for the persistence provider that the transaction must be read only the rollback rules for the transaction.*
5. ***@Transient***

*The @****Transient*** *is used to specify that a given entity attribute should not be persisted.*

1. ***@ UniqueConstraint***

*The @****UniqueConstraint*** *is used to specify a unique constraint to be included by the automated schema generator for the primary or secondary table associated with the currently annotated entity.*

1. ***@Version***

*The @****Version*** *is used to specify the version attribute used for optimistic locking.*

1. ***Spring Data Mongo Annotations***
2. ***@Document***

*The* ***@Document*** *marks a class as being a domain object that we want to persist to the database.*

1. ***@Field***

*The* ***@Field*** *can configure the name of a field we want to use when MongoDB persist the document.*

1. ***@Query***

*The* ***@Query*** *can provide a finder query on a MongoDB repository method.*

1. ***@EnableMongoRepositories***

*The* ***@EnableMongoRepositories*** *used to indicate Spring to use MongoDB repositories.*

1. ***Cache Based Annotations***
2. ***@Cacheable***

*The* ***@Cacheable*** *is used on methods. The simplest way of enabling the cache behaviour for a method is to annotate it with @Cacheable and parameterize it with the name of the cache where the result would be stored.*

1. ***@CachePut***

*The* ***@ChachePut*** *is used on methods. Whenever you need to update the cache without interfering the method execution, you can use the @CachePut annotation. Using @CachePut and @Cacheable on the same method is strongly discouraged as the former forces the execution a cache update, the latter causes the method execution to be skipped by using the cache.*

1. ***@CacheEvict***

*The* ***@CacheEvict*** *is used on methods. It is not that you always want to populate the cache with more and more data. Sometimes you may want remove some cache data so that you can populate the cache with some fresh values.*

1. ***@CacheConfig***

*The* ***@CacheConfig*** *is a class level annotation. The @CacheConfig annotation helps to streamline some of the cache information at one place. Placing this annotation on a class does not turn on any caching operations. This allows you to store the cache configuration at the class level so that you don’t have declare things multiple times.*

1. ***Task Execution And Scheduling Annotations***
2. ***@Scheduled***

*The* ***@Scheduled*** *is used on methods. The @Scheduled annotation is used on methods along with trigger metadata. A method with @Scheduled should have void return type and should not accept any parameters.*

1. ***@Async***

*The* ***@Async*** *is used on methods to execute each method in a separate thread. The @Async annotation is provided on a method so that the invocation of that method will occur asynchronously. Unlike methods annotated with @Scheduled, the methods annotated with @Async can take arguments. They will be invoked in the normal way by callers at runtime rather than by a scheduled task. @Async can be used with both void return type methods and the methods that return a value. However methods with return value must have a future typed return values.*

1. ***Spring Security Annotations***
2. ***@Secure***

*The* ***@Secure*** *annotation is used to specify list of roles on the method.*

1. ***@RolesAllowed***

*The* ***@RolesAllowed*** *annotation is the JSR-250’s equivalent annotation of the @Secured annotation. Basically, we can use this annotation in a similar way as @Secured.*

1. ***@PreAuthorize***

*The* ***@PreAuthorize*** *annotations checks the given expression before entering the method.*

1. ***@PostAuthorize***

*The* ***@PostAuthorize*** *annotation verifies it after the execution of the method and could alter the result. Additionally this annotation provides the ability to access the method result.*

1. ***@PreFilter***

*The* ***@PreFilter*** *annotation provides to filter a collection argument before executing the method.*

1. ***@PostFilter***

*The* ***@PostFilter*** *annotation provides to filter the returned collection of a method.*

1. ***Spring Framework Testing Annotations***
2. ***@BootstrapWith***

*The* ***@BootstrapWith*** *is a class level annotation. The annotation is used to configure how the Spring TestContext Framework is bootstrapped. This annotation is used a metadata to create custom composed annotations and reduce the configuration duplication in a test suite.*

1. ***@ContextConfiguration***

*The* ***@ContextConfiguration*** *is a class level annotation that defines a metadata used to determine which configuration files to use to the load the ApplicationContext for your test.*

1. ***@WebAppConfiguration***

*The* ***@WebAppConfiguration*** *is a class level annotation, is used to declare that the ApplicationContext loaded for an integration test should be a WebApplicationContext. This annotation is used to create the web version of the application context. Used with the @ContextConfiguration annotation.*

1. ***@Timed***

*The* ***@Timed*** *is used on methods, indicates that the annotated test method must finish its execution at the specified time period (in milliseconds). If the execution exceeds the specified time the annotation, the test fails.*

1. ***@Repeat***

*The* ***@Repeat*** *is used on test methods, used if you want to run a test method several times in a row automatically. The number of times that test method is to be executed is specified in the annotation.*

1. ***@Commit***

*The* ***@Commit*** *can be used as both class-level or method-level annotation, after execution of a test method, the transaction of the transactional test method can be commited using the @Commit annotation.*

1. ***@RollBack***

*The* ***@RollBack*** *can be used as both class-level or method-level annotation, indicates whether the transaction of a transactional test method must be rolled back after the test completes its execution.*

1. ***@DirtiesContext***

*The* ***@DirtiesContext*** *can be used as both class-level or method-level annotation, indicates that the Spring ApplicationContext has been modified or corrupted in some manner and it should be closed. This will trigger the context reloading before execution of next test. The ApplicationContext is marked as dirty before or after any such annotated method as well as before or after current test class. This annotation supports BEFORE\_METHOD, BEFORE\_CLASS, and BEFORE\_EACH\_TEST\_METHOD modes for closing the ApplicationContext before a test.*

1. ***@BeforeTransaction***

*The* ***@BeforeTransaction*** *is used to annotate void method in the class, indicate that they should be executed before any transaction starts executing. That means the method annotated must be executed before any method annotated with @Transactional.*

1. ***@AfterTransaction***

*The* ***@AfterTransaction*** *is used to annotate void method in the class, indicate that they should be executed after any transaction starts executing. That means the method annotated must be executed after any method annotated with @Transactional.*

1. ***@Sql***

*The* ***@Sql*** *can be declared on a test class or test method to run SQL script against a database. This annotation configures the resource path to SQL scripts that should be executed against a given database either before or after an integration test method. When this used at the method level it will override any @Sql defined in at class level.*

1. ***@SqlConfig***

*The* ***@SqlConfig*** *used along with the @Sql annotation, defines the metadata that is used to determine how to parse and execute SQL scripts configured via the @Sql.*

1. ***@SqlGroup***

*The* ***@SqlGroup*** *is used on methods. The @SqlGroup annotation is a container annotation that can hold several @Sql annotation. In addition , used as a meta-annotation to create custom composed annotations.*

1. ***@SpringBootTest***

*The* ***@SpringBootTest*** *used to start the Spring context for integration tests. This will bring up the full auto configuration context.*

1. ***@DataJpaTest***

*The* ***@DataJpaTest*** *will only provide the auto configuration required to test Spring Data JPA using in memory database such as H2. Used instead @SpringBootTest.*

1. ***@DataMongoTest***

*The* ***@DataMongoTest*** *will provide a minimal auto configuration and an embedded MongoDB for running integration tests with Spring Data MongoDB.*

1. ***@WebMVCTest***

*The* ***@WebMVCTest*** *will bring up a mock servlet context for testing MVC layer. Services and components are not loaded into the context. To provide these dependencies for testing, the @MockBean annotation is typically used.*

1. ***@AutoConfigureMockMVC***

*The* ***@AutoConfigureMockMVC*** *annotation works very similar to the @WebMVCTest annotation, but the full Spring Boot context is started.*

1. ***@MockBean***

*The* ***@MockBean*** *creates and inject a Mockito Mock for the given dependency.*

1. ***@JsonTest***

*The* ***@JsonTest*** *will limit the auto configuration of Spring Boot to components relevant to processing JSON. This annotation will also auto configure an instance of JacksonTester or GsonTester.*

1. ***@TestPropertySource***

*The* ***@TestPropertySource*** *is class level annotation, used to specify property sources for the test class.*

1. ***Spring Boot***