**Core Java Frequently Asked Questions**

**Java Fundamentals:**

1. **What is JVM and Is it platform independent?**

**Ans:** JVM is an acronym for Java Virtual Machine, it is an abstract machine which provides the runtime environment in which java bytecode can be executed. It is a specification.

JVMs are available for many hardware and software platforms (so JVM is platform dependent).

1. **What is difference between JDK and JVM and JRE?**

### Ans:

### JVM

JVM is an acronym for Java Virtual Machine, it is an abstract machine which provides the runtime environment in which java bytecode can be executed. It is a specification.

JVMs are available for many hardware and software platforms (so JVM is platform dependent).

**JRE**

JRE stands for Java Runtime Environment. It is the implementation of JVM.

**JDK**

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.

1. **Why Java is not pure Object Oriented language?**

**Ans**: Java is supporting primitive data type and wrapper object so Java is not pure object oriented language.

1. **What are principle concepts of OOPS?**

**Ans**: 1) Encapsulation

2)Abstraction

3)Inheritance

4)Polymorphism

1. **What is Java Package and which package is imported by default?**

**Ans:** Package is a collection of similar types of classes and methods. Default package is java.lang.

1. **What is Overloading and Overriding in Java?**

**Ans: Overloading**

If a class have multiple methods by same name but different parameters, it is known as Method Overloading. It increases the readability of the program.

**Overriding**

If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding. It is used for runtime polymorphism and to provide the specific implementation of the method

1. **What is the difference between an Inner Class and a Sub-Class?**

**Ans:** Inner class is non-static nested class. It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class do. Defining one class with in another class . It is known as nested class.

1. **What is the difference between abstract class and interface?**

**Ans:** • Interface cannot implement any methods, where as abstract class can implement(it can contain both declared and defined methods).

• A class can implement many interfaces but can have only one super class(abstract)

• Interface members are by default public

• Abstract class members are default

• Interface data fields are final where as abstract class data fields are not final

• Interface is null implemenation and abstract class is partial implementation

1. **What is static binding and dynamic binding?**

**Ans**: Static binding occurs during compile time and dynamic binding occurs during run time.

1. **What is Data Encapsulation and what’s its significance?**

**Ans**: A class can contain any number of data members the grouping up of data members and functions together is encapsulation. it is used to separate the essential entities from non essential data members and functions are encapsulated to form a family called class.

1. **What is Java Bean Class?**

**Ans**: It is an interface between jsp and database. It is used to write business logic methods. Java bean class should be kept with in the package. Java bean class should be having getter and setter methods with one constructor.

1. **What are Access modifiers?**

**Ans:** It specifies the accessibility of the members of the class.

These are the access modifiers :

default

private

public

protected

1. **What’s the benefit of using inheritance?**

**Ans**: We can use the base class properties in sub class its best practice for code reusability, its reduce re declaration and complexity of the program.

1. **Why multiple Inheritance not supported in Java?**

**Ans:** A class can implement any number of interfaces but can extend only one class.

1. **What is the diamond problem in inheritance?**

**Ans:** Multiple inheritance is not supported because it leads to deadly diamond problem.

1. **What is the difference between break and continue statement?**

**Ans:** When the break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

It causes the loop to immediately jump to the next iteration of the loop. In a for loop, the continue keyword causes control to immediately jump to the update statement.

1. **What is nested class?**

**Ans**: Class inside the class is called nested class.

1. **How are this() and super() used with Constructor?**

**Ans: this()**

Using the this Keyword. Within an instance method or a constructor, this is a reference to the current object — the object whose method or constructor is being called.

**super()**

super is a keyword. It is used inside a sub-class method definition to call a method defined in the super class. Private methods of the super-class cannot be called. Only public and protected methods can be called by the super keyword.

1. **What is Serialization and Deserialization**

**Ans:** Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.

1. **What is difference between Heap and Stack Memory?**

**Ans:** Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer's RAM. The basic difference between stack and heap is the life cycle of the values. Stack values only exist within the scope of the function they are created in. Once it returns, they are discarded. Heap values however exist on the heap. They are created at some point in time, and destructed at another either by Garbage Collector or manually.

1. **How garbage collection is done in Java?**

**Ans**: In garbage collection at the end of the program execution unreferred objects are going to delete in memory and free the memory area.

1. **How destructors are defined in Java?**

**Ans**: Destructors are not support in java it’s used in C++, in java we use garbage collection.

**Array:**

1. **What do you mean by an Array? How to create?**

**Ans: Array is data structure, it’s use to store similar types of data. It’s homogeneous data type.**

**There are two types of array**

1. **Single dimensional array.**
2. **Multi dimensional array.**

**Array declaration:**

**Data type [] arrayname;**

**Data type arrayname[];**

**Exam: int a[]=new a[10];**

1. **Advantages and disadvantages of Array?**

**Ans: Advantages:**

1. **Random access: we can access the data inside the array**
2. **Code optimization**

**Disadvantages:**

**Fixed size: it’s doesn’t grow randomly.**

1. **What is the meaning of anonymous array? Explain with an example?**

**Ans:** In array without name is known as anonymous array in java. As the array do not have any name so it can be used only once.  Anonymous array is passed as an argument of method.

new int[]{10,20,30,40}

1. **What are “jagged” arrays in java?**

**Ans:** The elements of a jagged array can be of different dimensions and sizes. A jagged array is sometimes called an array of arrays.

1. **How to copy an array into another array?**

**Ans:** We can copy an array to another by the [array](https://www.javatpoint.com/array-in-java)copy method of System class.

**Syntax:** public static void arraycopy(Object src, int srcPos,Object dest, int destPos, int length  )

1. **What is the step to access elements of an array in Java**

**Ans: Using index of an Array.**

**Ex: a[i]; // i is index value.**

1. **What is difference between ArrayIndexOutfOBounds and ArrayStoreException?**

**Ans:** At the time of initialization of array we initialize the size or capacity of the array, if we try to insert the data out of that capacity we’ll get ArrayIndexOutOfBound Exception.

  ArrayStoreException exception shows when we try to insert other data type element to array.

1. **How to check array contains value or not?**

**Ans: By contain() method or using length() method.**

1. **How to check array contains value or not?**

**Ans:** Using Contains() method.

1. **Where does array stored in memory?**

**Ans**: A statically-created array will be on the stack. A dynamically-created array (with new or malloc ) will be allocated on the heap.

**String:**

1. **What is String in Java?**

**Ans**: String is a Class in java and defined in java.lang package. String class represents character Strings. String in immutable and final in Java and JVM uses String Pool to store all the String objects.

1. **What are different ways to create String Object?**

**Ans:** By literal

String a=”ChanduPallu”;

By New keyword

String a= new String(“ChanduPallu”);

1. **What is String subSequence method?**

**Ans:** subsequence method is internally invoke substring() method, it’ll select some part of the string.

Substring(int beginIndex, int endIndex).

1. **How to convert String to char and vice versa?**

**Ans:** String to char:

By charAt() method

String a=”Chandu”;

Char b=a.charAt(0);// b=c;

Char to string:

1. By String.valueOf() method

Char a= ‘c’;

String b=Sttring.valueOf(c);

1. By toString();

Char a=’c’;

String b=character.toString();

1. **How to convert String to byte array and vice versa?**

**Ans:** We can use String getBytes() method to convert String to byte array and we can use String constructor new String(byte[] arr) to convert byte array to String.

1. **Difference between String, StringBuffer and StringBuilder?**

**Ans:** **String:** String class is immutable.String is slow and consumes more memory when you concat too many strings because every time it creates new instance.

**StringBuffer:** [StringBuffer](https://www.javatpoint.com/difference-between-string-and-stringbuffer) class is mutable. StringBuffer is synchronized. StringBuffer is less efficient than StringBuilder.

**StringBuilder:** [StringBuffer](https://www.javatpoint.com/difference-between-string-and-stringbuffer) class is mutable. StringBuilder is non-synchronized. StringBuilder is more efficient than StringBuffer.

1. **Why String is immutable or final in Java?**

**Ans:** It increases security because any hacker can’t change its value and it’s used for storing sensitive information such as database username, password etc. Since String is immutable, it’s safe to use in multi-threading and we don’t need any synchronization.

1. **What is String Pool?**

**Ans:** String Pool in java is a pool of Strings stored in Java Heap Memory. If create string using new keyword that string will going to store in string pool.

1. **What does String intern() method do?**

**Ans:** Java's String class privately maintains a pool of strings, where String literals are automatically interned. When the intern() method is invoked on a String object it looks the string contained by this String object in the pool, if the string is found there then the string from the pool is returned.

1. **Why String is popular HashMap key in Java?**

**Ans:** Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for key in a Map and its processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

**Collections:**

1. **What is Java Collections Framework? List out some benefits of Collections framework?**

**Ans:** Collections framework is a framework of framework, it’ll provide framework support for Hibernate, Jpa etc..

Collection solves the disadvantages of Array, Collection is used for storage purpose. In array we facing fixed size declaration and it’s homogeneous but collection is a Heterogeneous and random-access, auto resizing.

1. **What is the benefit of Generics in Collections Framework?**

**Ans:** Collocation is heterogeneous also we specifies the data type using generics method because for validation of data and it’ll be use for fast compilation as well as it’s use full for type casting.

1. **What are the basic interfaces of Java Collections Framework?**

**Ans:** Collection is base interface, in Collections framework basically we use 9 interfaces collection, list, set, queue, map, sorted set, navigable set, sorted map and navigable map.

1. **What are common algorithms implemented in Collections Framework?**

**Ans:** Collection is used for storing data so based on data we perform some operation for that operation we use Sorting algorithm and searching algorithm.

1. **Why Collection doesn’t extend Cloneable and Serializable interfaces?**

**Ans:** Collection is an interface that specifies a group of objects known as elements. The details of how the group of elements is maintained is left up to the concrete implementations of Collection.

**For example:** some Collection implementations like List allow duplicate elements whereas other implementations like Set don't. A lot of the Collection implementations have a public clone method. However, it does't really make sense to include it in all implementations of Collection. This is because Collection is an abstract representation. What matters is the implementation. The semantics and the implications of either cloning or serializing come into play when dealing with the actual implementation; that is, the concrete implementation should decide how it should be cloned or serialized, or even if it can be cloned or serialized. In some cases, depending on what the actual backing-implementation is, cloning and serialization may not make much sense. So mandating cloning and serialization in all implementations is actually less flexible and more restrictive.

1. **What is difference between Array and ArrayList? When will you use Array over ArrayList?**

**Ans:**

|  |  |
| --- | --- |
| Array | ArrayList |
| 1. It’s Fixed size, declared in initialization part. | 1. Grow able, it’s grow as per requirements. |
| 1. Fast process. | 1. Compare to array little bit slow. |
| 1. Homogeneous collection of data. | 1. Heterogeneous collection of data. |
| 1. It’s don’t have ready made algorithm, or standard methods. | 1. It’s have standard menthods. |
| 1. It’s hold primitive as well as object also. | 1. Its hold only object. |

1. **What are similarities and difference between ArrayList, Linked List and Vector?**

**Ans:**

|  |  |  |
| --- | --- | --- |
| Linked List | ArrayList | Vector |
| Non synchronize | Non synchronize | Synchronized |
| Relatively performance is very low for retrieving operation | Relatively performance is high | Relatively performance is low |
| At a time multithread also use, no thread safe | At a time multithread also use, no thread safe | Only allowed single thread, thread safe |
| Released on 1.2 version no legacy class | Released on 1.2 version no legacy class | Released on 1.0 version its legacy class |
| Use double linked list structure | Use Array data structure | Use double linked list data structure |

1. **Why Map interface doesn’t extend Collection interface?**

**Ans:** Collection objects that contains only values, based on that values we’ll perform operations and data structure is also difference compared to map. In map interface the data structure stored as key-value pair. Based on key only we have to perform operations.

1. **What is difference between Enumeration and Iterator interface?**

**Ans:**

|  |  |
| --- | --- |
| Enumeration | Iterator |
| Enumeration concepts Is applied only for legacy class. | Enumeration concepts Is applied for all class. |
| It’s nota universal cursor. | It’s universal cursor. |
| We can perform only read operation. | We can perform read and remove operation also. |

1. **What is difference between Stack and Queue?**

**Ans:**

|  |  |
| --- | --- |
| Stack | Queue |
| LIFO (Last in First out) | FIFO (First in First out) |
| Same end is used to insert and delete elements. | One end is used for insertion, i.e., rear end and another end is used for deletion of elements, i.e., front end. |
| Operation performed Push and Pop | Operation performed Enqueue and dequeue |

1. **What is difference between Comparable and Comparator interface?**

**Ans:**

|  |  |
| --- | --- |
| Comparable | Comparator |
| Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc. | Comparator provides **multiple sorting sequence.** In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc. |
| Comparable **affects the original class** i.e. actual class is modified. | Comparator **doesn't affect the original class** i.e. actual class is not modified. |
| provides **compareTo() method** to sort elements. | provides **compare() method** to sort elements. |
| found in **java.lang** package. | is found in **java.util** package. |
| We can sort the list elements of [Comparable](https://www.javatpoint.com/difference-between-comparable-and-comparator) type by [Collection](https://www.javatpoint.com/difference-between-comparable-and-comparator)**s.sort(List)** method. | We can sort the list elements of Comparator type by [Collection](https://www.javatpoint.com/difference-between-comparable-and-comparator)**s.sort(List,**[Comparator](https://www.javatpoint.com/difference-between-comparable-and-comparator)**)** method. |

1. **How HashMap works in Java?**

**Ans:** A HashMap contains values based on the key. It contains only unique elements. It may have one null key and multiple null values. It maintains no order.

1. **What are different Collection views provided by Map interface?**

**Ans:** It provides **three collection views** methods which allow Map to be viewed as Collection in below ways:

* **Keyset**: This method provides Set of the keys contained in Map.
* **entrySet**: This method provides Collection of values contained in Map. Please note basically this Collection is not Set because we can have multiple keys that is map to same value.
* **Values**: This method provides Set of key value pairs contained in Map. Map interface provides small nested interface that is called Map.Entry and type of the elements in this Set.

**Exception Handling:**

1. **What is Exception &Exception Handling in Java?**

**Ans:** Exception is an error event that can happen during the execution of a program and disrupts its normal flow. Java provides a robust and object oriented way to handle exception scenarios, known as **Java Exception Handling**.

1. **What are the Exception Handling Keywords in Java?**

**Ans:**  Java provides specific keywords for exception handling purposes,

1. **throw** – We know that if any exception occurs, an exception object is getting created and then Java runtime starts processing to handle them. Sometime we might want to generate exception explicitly in our code, for example in a user authentication program we should throw exception to client if the password is null. **throw** keyword is used to throw exception to the runtime to handle it.
2. **throws** – When we are throwing any exception in a method and not handling it, then we need to use **throws** keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to it’s caller method using throws keyword. We can provide multiple exceptions in the throws clause and it can be used with [main()](https://www.journaldev.com/611/exception-in-thread-main-java) method also.
3. **try-catch** – We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.
4. **finally** – finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurred or not.
5. Explain Java Exception Hierarchy?

**Ans:** when any exception is raised an **exception object** is getting created. Java Exceptions are hierarchical and [inheritance](https://www.journaldev.com/644/inheritance-java-example) is used to categorize different types of exceptions. Throwable is the parent class of Java Exceptions Hierarchy and it has two child objects – Error and Exception. Exceptions are further divided into checked exceptions and runtime exception.

1. **Errors**: Errors are exceptional scenarios that are out of scope of application and it’s not possible to anticipate and recover from them, for example hardware failure, JVM crash or out of memory error. That’s why we have a separate hierarchy of errors and we should not try to handle these situations. Some of the common Errors are OutOfMemoryError and StackOverflowError.
2. **Checked Exceptions**: Checked Exceptions are exceptional scenarios that we can anticipate in a program and try to recover from it, for example FileNotFoundException. We should catch this exception and provide useful message to user and log it properly for debugging purpose. Exception is the parent class of all Checked Exceptions and if we are throwing a checked exception, we must catch it in the same method or we have to propagate it to the caller using throws keyword.
3. **Runtime Exception**: Runtime Exceptions are cause by bad programming, for example trying to retrieve an element from the Array. We should check the length of array first before trying to retrieve the element otherwise it might throw ArrayIndexOutOfBoundException at runtime. RuntimeException is the parent class of all runtime exceptions. If we are throwing any runtime exception in a method, it’s not required to specify them in the method signature throws clause. Runtime exceptions can be avoided with better programming.
4. **What are important methods of Java Exception Class?**

**Ans:**

1. **public String getMessage()** – This method returns the message String of Throwable and the message can be provided while creating the exception through it’s constructor.
2. **public String getLocalizedMessage()** – This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use getMessage() method to return the exception message.
3. **public synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.
4. **public String toString()** – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.
5. **public void printStackTrace()** – This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as argument to write the stack trace information to the file or stream.
6. **What is difference between Checked and Unchecked Exception in Java?**

**Ans:**

* 1. **Checked Exception**

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

* 1. **Unchecked Exception**

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

1. **What is difference between throw and throws keyword in Java?**

**Ans:**

|  |  |
| --- | --- |
| Throw | throws |
| Java throw keyword is used to explicitly throw an exception. | Java throw keyword is used to explicitly throw an exception. |
| Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| Throw is followed by an instance. | Throws is followed by class. |
| Throw is used within the method. | Throws is used with the method signature. |
| You cannot throw multiple exceptions. | You can declare multiple exceptions e.g. public void method()throws IOException,SQLException. |

1. **How to write custom exception in Java?**

**Ans:** If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

Let's see a simple example of java custom exception.

class InvalidAgeException extends Exception{

 InvalidAgeException(String s){

  super(s);

 }

}

class TestCustomException1{

   static void validate(int age)throws InvalidAgeException{

     if(age<18)

      throw new InvalidAgeException("not valid");

     else

      System.out.println("welcome to vote");

   }

   public static void main(String args[]){

      try{

      validate(13);

      }catch(Exception m){System.out.println("Exception occured: "+m);}

      System.out.println("rest of the code...");

  }

}

1. **What are different scenarios causing “Exception in thread main”?**

**Ans:** Some of the common main thread exception scenarios are:

* **Exception in thread main java.lang.UnsupportedClassVersionError**: This exception comes when your java class is compiled from another JDK version and you are trying to run it from another java version.
* **Exception in thread main java.lang.NoClassDefFoundError**: There are two variants of this exception. The first one is where you provide the class full name with .class extension. The second scenario is when Class is not found.
* **Exception in thread main java.lang.NoSuchMethodError: main**: This exception comes when you are trying to run a class that doesn’t have main method.
* **Exception in thread “main” java.lang.ArithmeticException**: Whenever any exception is thrown from main method, it prints the exception is console. The first part explains that exception is thrown from main method, second part prints the exception class name and then after a colon, it prints the exception message.

1. **What is difference between final, finally and finalize in Java?**

**Ans:** final and finally are keywords in java whereas finalize is a method.

final keyword can be used with class variables so that they can’t be reassigned, with class to avoid extending by classes and with methods to avoid overriding by subclasses, finally keyword is used with try-catch block to provide statements that will always gets executed even if some exception arises,

usually finally is used to close resources. finalize() method is executed by Garbage Collector before the object is destroyed, it’s great way to make sure all the global resources are closed.

Out of the three, only finally is related to java exception handling.

1. **What happens when exception is thrown by main method?**

**Ans:** When exception is thrown by main() method, Java Runtime terminates the program and print the exception message and stack trace in system console.

**Annotations:**

1. **What are annotations? What are their typical use cases?**

**Ans:** Annotations are metadata bound to elements of the source code of a program and have no effect on the operation of the code they operate.

Their typical uses cases are:

* **Information for the compiler** – with annotations, the compiler can detect errors or suppress warnings
* **Compile-time and deployment-time processing** – software tools can process annotations and generate code, configuration files, etc.
* **Runtime processing** – annotations can be examined at runtime to customize the behavior of a program

1. **Describe some useful annotations from the standard library.**

**Ans:** There are several annotations in the java.lang and java.lang.annotation packages, the more common ones include but not limited to:

* **@Override** *–* marks that a method is meant to override an element declared in a superclass. If it fails to override the method correctly, the compiler will issue an error
* **@Deprecated**– indicates that element is deprecated and should not be used. The compiler will issue a warning if the program uses a method, class, or field marked with this annotation
* **@SuppressWarnings**– tells the compiler to suppress specific warnings. Most commonly used when interfacing with legacy code written before generics appeared
* **@FunctionalInterface** – introduced in Java 8, indicates that the type declaration is a functional interface and whose implementation can be provided using a Lambda Expression

1. **How can you create an annotation?**

**Ans:** Annotations are a form of an interface where the keyword *interface* is preceded by *@,* andwhose body contains *annotation type element* declarations that look very similar to methods:

|  |  |
| --- | --- |
| 1  2  3  4  5 | public @interface SimpleAnnotation {      String value();        int[] types();  } |

After the annotation is defined, yon can start using it in through your code:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @SimpleAnnotation(value = "an element", types = 1)  public class Element {      @SimpleAnnotation(value = "an attribute", types = { 1, 2 })      public Element nextElement;  } |

Note that, when providing multiple values for array elements, you must enclose them in brackets.

Optionally, default values can be provided as long as they are constant expressions to the compiler:

|  |  |
| --- | --- |
| 1  2  3  4  5 | public @interface SimpleAnnotation {      String value() default "This is an element";      int[] types() default { 1, 2, 3 };  } |

Now, you can use the annotation without those elements:

|  |  |
| --- | --- |
| 1  2  3  4 | @SimpleAnnotation  public class Element {      // ...  } |

Or only some of them:

|  |  |
| --- | --- |
| 1  2 | @SimpleAnnotation(value = "an attribute")  public Element nextElement; |

1. **What object types can be returned from an annotation method declaration?**

**Ans:** The return type must be a primitive, String, Class, Enum, or an array of one of the previous types. Otherwise, the compiler will throw an error.

Here’s an example code that successfully follows this principle:

enum Complexity {

    LOW, HIGH

}

public @interface ComplexAnnotation {

    Class<? extends Object> value();

    int[] types();

    Complexity complexity();

}

The next example will fail to compile since Object is not a valid return type:

public @interface FailingAnnotation {

    Object complexity();

}

1. **Which program elements can be annotated?**

**Ans:** Annotations can be applied in several places throughout the source code. They can be applied to declarations of classes, constructors, and fields:

@SimpleAnnotation

public class Apply {

    @SimpleAnnotation

    private String aField;

    @SimpleAnnotation

    public Apply() {

        // ...

    }

}

Methods and their parameters:

@SimpleAnnotation

public void aMethod(@SimpleAnnotation String param) {

    // ...

}

Local variables, including a loop and resource variables:

@SimpleAnnotation

int i = 10;

for (@SimpleAnnotation int j = 0; j < i; j++) {

    // ...

}

try (@SimpleAnnotation FileWriter writer = getWriter()) {

    // ...

} catch (Exception ex) {

    // ...

}

Other annotation types:

@SimpleAnnotation

public @interface ComplexAnnotation {

    // ...

}

And even packages, through the package-info.java file:

@PackageAnnotation

package com.[baeldung](http://www.baeldung.com/java-annotations-interview-questions).interview.annotations;

As of Java 8, they can also be applied to the *use* of types. For this to work, the annotation must specify an *@Target* annotation with a value of *ElementType.USE*:

|  |
| --- |
|  |

@Target(ElementType.TYPE\_USE)

public @interface SimpleAnnotation {

    // ...

}

Now, the annotation can be applied to class instance creation: new @SimpleAnnotation Apply();

Type casts:

aString = (@SimpleAnnotation String) something;

Implements clause:

public class SimpleList<T>

  implements @SimpleAnnotation List<@SimpleAnnotation T> {

    // ...

}

And throws clause:

void aMethod() throws @SimpleAnnotation Exception {

    // ...

}

1. **What are meta-annotations?**

**Ans:** Are annotations that apply to other annotations.

All annotations that aren’t marked with @Target, or are marked with it but include ANNOTATION\_TYPE constant are also meta-annotations:

@Target(ElementType.ANNOTATION\_TYPE)

public @interface SimpleAnnotation {

    // ...

}

1. **What are repeating annotations?**

**Ans:** These are annotations that can be applied more than once to the same element declaration.

For compatibility reasons, since this feature was introduced in Java 8, repeating annotations are stored in a container annotation that is automatically generated by the Java compiler. For the compiler to do this, there are two steps to declared them.

First, we need to declare a repeatable annotation:

@Repeatable(Schedules.class)

public @interface Schedule {

    String time() default "morning";

}

Then, we define the containing annotation with a mandatory value element, and whose type must be an array of the repeatable annotation type:

public @interface Schedules {

    Schedule[] value();

}

Now, we can use @Schedule multiple times:

@Schedule

@Schedule(time = "afternoon")

@Schedule(time = "night")

void scheduledMethod() {

}

**Regex**

**1.What is regex? why we go for regex?**

**Ans:** The Java Regex or Regular Expression is an API to define pattern for searching or manipulating strings.

It is widely used to define constraint on strings such as password and email validation. We will be able to test our own regular expressions by the Java Regex Tester Tool.

**02) What are the classes in Java that helps to deal with regular expressions?**

**Ans:** Java has a dedicated package named java.util.regex that has three classes which help to deal with regular expressions. Following is a brief description about them.

**Pattern** – represents compiled representation of a regex. You can get a new instance by using the static ‘compile’ method which accepts a regular expression as the first argument.

**PatternSyntaxException** – unchecked exception that occurs when there is a problem with the regular expression pattern’s syntax.

**Matcher** – engine that interprets the pattern and does match operations for an input string. You can get a new instance by using a Pattern object’s matcher method.

**03)** **What is a metacharacter?**

**Ans:** A metacharacter is a character that has a special meaning to a regular expression engine. This will not be counted as a regular character by the regex engine. Examples of metacharacters are ^, $, ., \*, +, etc.

**04)What are predefined character classes?**

**Ans:** Predefined character classes are useful shorthand notations available for commonly used regular expressions.

Predefined Character Class Description

. Any character

d 0-9

s Whitespace character

w A word character, ie [A-Za-z\_0-9]

D Non-digit character

W Non-word character

S Non-whitespace character

**05) Which is regex engine class?**

**Ans:** Matcher class is the engine of regex and it implements MatchResult(I), used perform matching operations on a character sequence.

**06) Which is compiler of regex?**

**Ans:** Pattern class the compiled version of regex, used to define a pattern for the regex engine.

**07) Advantages of regex?**

**Ans:** With smart code completion, safe refactoring,and better support for Node.js, angular.

**ENUM**

**01) What is enum and why we go for enum?**

**Ans:** It is a data type that contains fixed set of constants.

It can be used for days of the week , directions etc. The java enum constants are static and final implicitly. It is available from JDK 1.5.

**02) Can Enum implement interface in Java?**

**Ans:** Yes, Enum can implement interface in Java. Since enum is a type, similar to class and interface, it can implement interface. This gives a lot of flexibility to use Enum as specialized implementation in some cases.

**03) Can Enum extends class in Java?**

**Ans:** No, Enum can not extend class in Java. Since all Enum by default extend abstract base class java.lang.Enum, obviously they can not extend another class, because Java doesn't support multiple inheritance for classes. Because of extending java.lang.Enum class, all enum gets methods like ordinal(), values() or valueOf().

**04) Can we declare Constructor inside Enum in Java?**

**Ans:** Yes, you can, but remember you can only declare either private or package-private constructor inside enum. public and protected constructors are not permitted inside enum.

**05) Can we override toString() method for Enum? What happens if we don't?**

**Ans:** Yes we can override toString in Enum, as like any other class it also extends java.lang.Object and has toString()method available, but even if you don't override, you will not going to regret much, because abstract base class of enum does that for you and return name, which is name of the enum instance itself. here is the code of toString() method from Enum class

**06) How do you create Enum without any instance?**

**Ans:** yes, we can create Enum without any instance in Java, say for creating a utility class.