**Immutable**

1. **Does Declaring an object "final" makes it immutable?**

Only declaring primitive types as final makes them immutable. Making objects final means that the object handler cannot be used to target some other object but the object is still mutable.

1. **Why Char array is preferred over String for storing password?**

String is immutable in java and stored in String pool. Once it’s created it stays in the pool until unless garbage collected, so even though we are done with password it’s available in memory for longer duration and there is no way to avoid it. It’s a security risk because anyone having access to memory dump can find the password as clear text.

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

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.

1. **Why is String immutable in Java?**
2. **String Pool**

When a string is created and if the string already exists in the pool, the reference of the existing string will be returned, instead of creating a new object. If string is not immutable, changing the string with one reference will lead to the wrong value for the other references.

1. **To Cache its Hashcode**

If string is not immutable, one can change its hashcode and hence not fit to be cached.

1. **Security**

String is widely used as parameter for many java classes, e.g. network connection, opening files, etc. Making it mutable might possess threats due to interception by the other code segment.

<https://www.programcreek.com/2013/04/why-string-is-immutable-in-java/>

1. **Immutable classes and how to write own immutable class**

<https://www.journaldev.com/129/how-to-create-immutable-class-in-java>

Immutable class are the class, whose object cannot be modified once created

**To create an immutable class in java, you have to do following steps.**

1. Declare the class as final so it can’t be extended.

(if you don’t declare class as final you need to make sure each and every method of class is declared as final ) class as final strong immutability , method as final weak immutability

1. Make all fields private so that direct access is not allowed.
2. Don’t provide setter methods for variables
3. Make all mutable fields final so that its value can be assigned only once.
4. Initialize all the fields via a constructor performing deep copy. (so that even if the actual assign changes clone variable don’t change) ex hashmap as member variable
5. Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference. (so that the variable cannot be changed) ex hashmap as member variable

**Benefits**

1. Immutable objects are by default thread safe, can be shared without synchronization in concurrent environment.
2. Immutable object simplifies development, because its easier to share between multiple threads without external synchronization.
3. Immutable object boost performance of Java application by reducing synchronization in code.
4. Another important benefit of Immutable objects is reusability, you can cache Immutable object and reuse them, much like String literals and Integers. You can use static factory methods to provide methods like valueOf(), which can return an existing Immutable object from cache, instead of creating a new one.

Apart from above advantages, immutable object has disadvantage of creating garbage as well. Since immutable object cannot be reused and they are just a use and throw. String being a prime example, which can create lot of garbage and can potentially slow down application due to heavy garbage collection, but again that's extreme case and if used properly Immutable object adds lot of value.

**Exception**

1. **What is an exception?**

Exception is an abnormal condition which occurs during the execution of a program and disrupts normal flow of the program. This exception must be handled properly. If it is not handled, program will be terminated abruptly.

1. **What is the difference between error and exception in java?**

Errors are mainly caused by the environment in which an application is running. For example, OutOfMemoryError happens when JVM runs out of memory. Whereas exceptions are mainly caused by the application itself. For example, NullPointerException occurs when an application tries to access null object.

1. **Can we keep other statements in between try, catch and finally blocks?** – No
2. **Can we write only try block without catch and finally blocks?** -- No.
3. **What is unreachable catch block error?**

When you are keeping multiple catch blocks, the order of catch blocks must be from most specific to most general ones. i.e sub classes of Exception must come first and super classes later. If you keep super classes first and sub classes later, compiler will show unreachable catch block error.

1. **What are run time exceptions in java. Give example?**

The exceptions which occur at run time are called as run time exceptions. These exceptions are unknown to compiler. All sub classes of java.lang.RunTimeException and java.lang.Error are run time exceptions. These exceptions are unchecked type of exceptions. For example, NumberFormatException, NullPointerException, ClassCastException, ArrayIndexOutOfBoundException, StackOverflowError etc.

1. **What is OutOfMemoryError in java?**

OutOfMemoryError is the sub class of java.lang.Error which occurs when JVM runs out of memory.

1. **What are checked and unchecked exceptions in java?**

Checked exceptions are the exceptions which are known to compiler. These exceptions are checked at compile time only. Hence the name checked exceptions. These exceptions are also called compile time exceptions. Because, these exceptions will be known during compile time.

Unchecked exceptions are those exceptions which are not at all known to compiler. These exceptions occur only at run time. These exceptions are also called as run time exceptions. All sub classes of java.lang.RunTimeException and java.lang.Error are unchecked exceptions.

1. **Can we write code after finally block ?** -- No. it shows unreachable code error.
2. **How do you create customized exceptions in java?**

create class extending Exception class. Have a constructor and call super class constructor

1. **What is ClassCastException in java?**

ClassCastException is a RunTimeException which occurs when JVM unable to cast an object of one type to another type

1. **Can we override a super class method which is throwing an unchecked exception with checked exception in the sub class?**

No. If a super class method is throwing an unchecked exception, then it can be overridden in the sub class with same exception or any other unchecked exceptions but can not be overridden with checked exceptions.

1. **Which class is the super class for all types of errors and exceptions in java?** -- java.lang.Throwable
2. **Difference Between ClassNotFoundException Vs NoClassDefFoundError In Java**

ClassNotFoundException : class.forName("Student") and student.class is not present

NoClassDefFoundError : .class was present at compile time but some how remove in runtime

<https://www.geeksforgeeks.org/classnotfoundexception-vs-noclassdeffounderror-java/>

<https://javarevisited.blogspot.com/2011/07/classnotfoundexception-vs.html>

1. **Overriding and Exception handling**

Problem 1: If The SuperClass doesn’t declare an exception:

Case 1: If SuperClass doesn’t declare any exception and subclass declare checked exception -- compilation error

Case 2: If SuperClass doesn’t declare any exception and SubClass declare Unchecked exception -- works fine

Problem 2: If The SuperClass declares an exception

Case 1: If SuperClass declares an exception and SubClass declares exceptions other than the child exception of the SuperClass declared Exception -- compilation error

Case 2: If SuperClass declares an exception and SubClass declares an child exception of the SuperClass declared Exception. -- works fine

Case 3: If SuperClass declares an exception and SubClass declares without exception. -- works fine

1. **Interface used in try with resource** --> <https://www.baeldung.com/java-try-with-resources>

- The resources declared must implement the AutoCloseable interface.

- A Custom Resource with AutoCloseable

To construct a custom resource that will be correctly handled by a try-with-resources block, the class should implement the Closeable or AutoCloseable interfaces, and override the close method:

public class MyResource implements AutoCloseable {

@Override

public void close() throws Exception {

System.out.println("Closed MyResource");

}

}

- Resource Closing Order --> Resources that were defined/acquired first will be closed last

1. **How to have user defined runtime exception**

<https://www.mkyong.com/java/java-custom-exception-examples/>

public class UserRunTimeException extends RuntimeException //for unchecked exception

{

public UserRunTimeException (String message) {

super(message);

} //ex: if list size is greater than 50 throw above exception

}

public class UserDefinedException extends Exception //for checked exception

{

public UserDefinedException (String message) {

super(message);

}

}

1. **Top of Exception Hierarchy** - Throwable class
2. **handle exception using spring or spring boot**

<https://www.baeldung.com/exception-handling-for-rest-with-spring>

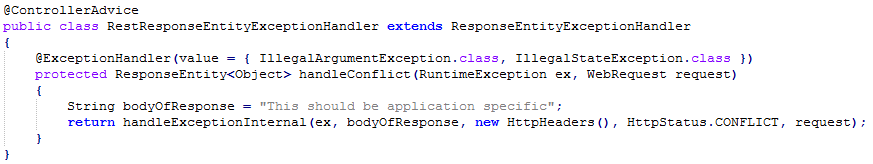
The @ControllerAdvice is an annotation, to handle the exceptions globally.

The @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.

**Before Spring 3.2, the two main approaches to handling exceptions in a Spring MVC application were:***HandlerExceptionResolver***or the***@ExceptionHandler***annotation.** Both of these have some clear downsides.

**Since 3.2 we've had the***@ControllerAdvice***annotation** to address the limitations of the previous two solutions and to promote a unified exception handling throughout a whole application.

Now, **Spring 5 introduces the***ResponseStatusException***class**: A fast way for basic error handling in our REST APIs

****

1. **Unchecked exception in inheritance**

**Rule**:  An overriding method (the method of child class) can throw any [unchecked exceptions](https://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/), regardless of whether the overridden method (method of base class) throws exceptions or not. However the overriding method should not throw [checked exceptions](https://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/) that are new or broader than the ones declared by the overridden method. The overriding method can throw those checked exceptions, which have less scope than the exception(s) declared in the overridden method.

* If base class doesn’t throw any exception but child class throws an unchecked exception. -- works fine
* If base class doesn’t throw any exception but child class throws an checked exception -- compilation error
* When base class and child class both throws a checked exception -- works fine
* When child class method is throwing border checked exception compared to the same method of base class -- compilation error

1. **How to handle "When an exception is thrown somewhere in the WHOLE program, exit."**

Thread.setDefaultUncaughtExceptionHandler(new Thread.UncaughtExceptionHandler()

{

public void uncaughtException(Thread t, Throwable e) {

LoggerFactory.getLogger("CustomLogger").error("Uncaught Exception in thread '" + t.getName() + "'", e);

System.exit(1);

}

});

1. **Can we write only try block without catch and finally blocks?**

No, It shows compilation error. The try block must be followed by either catch or finally block. You can remove either catch block or finally block but not both.

1. **What is unreachable catch block error?**

When you are keeping multiple catch blocks, the order of catch blocks must be from most specific to most general ones. i.e sub classes of Exception must come first and super classes later. If you keep super classes first and sub classes later, compiler will show unreachable catch block error.

1. **Can we keep the statements after finally block If the control is returning from the finally block itself?**

No, it gives unreachable code error. Because, control is returning from the finally block itself. Compiler will not see the statements after it. That’s why it shows unreachable code error.

1. **What is ClassCastException in java?**

ClassCastException is a RunTimeException which occurs when JVM unable to cast an object of one type to another type.

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

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

1. **What is Exception chaining in Java?**

Exception chaining is a popular exception handling concept in Java, where another exception is thrown in response of an exception and creating a chain of Exceptions. This technique mostly used to wrap a checked exception into an unchecked or RuntimeException. By the way if you are throwing new exception due to another exception then always include original exception so that handler code can access root cause by using methods like getCause() and initCause().

1. **Exception handling? How Exception is propagated in inheritance tree?**

<https://www.geeksforgeeks.org/exception-propagation-java/>

When an exception happens, Propagation is a process in which the exception is being dropped from to the top to the bottom of the stack. If not caught once, the exception again drops down to the previous method and so on until it gets caught or until it reach the very bottom of the call stack. **This is called exception propagation and this happens in case of Unchecked Exceptions only**.

Unlike Unchecked Exceptions, the propagation of exception **does not happen** in case of Checked Exception and its mandatory to use [**throw** keyword](https://www.geeksforgeeks.org/throw-throws-java/) here. Only unchecked exceptions are propagated.**Checked exceptions throw compilation error.**

1. **Global Exception Handler**
2. setDefaultUncaughtExceptionHandler method of thread
3. @ControllerAdvice

The instances of the RuntimeException are optional to handle. Consequently, it still leaves a window open for getting the long stack traces at runtime. To handle this, Java provides the UncaughtExceptionHandler interface. The Thread class contains this as an inner class.

In addition to this interface, Java 1.5 release also introduced a static method setDefaultUncaughtExceptionHandler() in the Thread class. The argument of this method is a handler class that implements the UncaughtExceptionHandler interface.

Furthermore, this interface declares the method uncaughtException(Thread t, Throwable e). It will be invoked when a given thread t terminates due to the given uncaught exception e. The implementing class implements this method and defines the logic for handling these uncaught exceptions.

Let's consider the following example that throws an ArithmeticException at runtime. We define the class Handler that implements the interface UncaughtExceptionHandler.

This class implements the method uncaughtException() and defines logic to handle uncaught exceptions in it:

public class GlobalExceptionHandler {

public static void main(String[] args) {

Handler globalExceptionHandler = new Handler();

Thread.setDefaultUncaughtExceptionHandler(globalExceptionHandler);

new GlobalExceptionHandler().performArithmeticOperation(10, 0);

}

public int performArithmeticOperation(int num1, int num2) {

return num1/num2;

}

}

class Handler implements Thread.UncaughtExceptionHandler {

private static Logger LOGGER = LoggerFactory.getLogger(Handler.class);

public void uncaughtException(Thread t, Throwable e) {

LOGGER.info("Unhandled exception caught!");

}

}

Here, the currently executing thread is the main thread. Thus, its instance is passed to the method uncaughtException() along with the raised exception. The class Handler then handles this exception.

