Method overloading:

* + May or may not have different return types.
  + Have different access modifiers.
  + Throw different checked or unchecked exceptions.

In the case of method overloading, the compiler decides which method is going to get called based on the reference upon which it is getting called and the method name, return type, and argument list.

when you extend a class, the subclass have all the methods defined by superclass. **It is as if all the methods of superclass have been implemented by the subclass.** That means the hello() method has been implemented by the class C as well. Now, you added a method in class C with different parameter (hello(String s)). That means, class C has two methods in all with same name but different parameters and that is "**overloading**".

Method overriding:

Method overriding means defining a method in a child class that is already defined in the parent class with the same method signature — same name, arguments, and return type (after Java 5, you can also use a covariant type as the return type).

Covariant return type refers to **return type of an overriding method**. It allows to narrow down return type of an overridden method without any need to cast the type or check the return type. Covariant return type works only for non-primitive return types.

Example of covariant type:

class SuperClass {  
   SuperClass get() {  
      System.out.println("SuperClass");  
      return this;  
   }  
}  
public class Tester extends SuperClass {  
   Tester get() {  
      System.out.println("SubClass");  
      return this;  
   }  
   public static void main(String[] args) {  
      SuperClass tester = new Tester();  
      tester.get();  
   }  
}

Mammal mammal = new Cat();

System.out.println(mammal.speak());

At the line mammal.speak(), the compiler says the speak() method of reference type Mammal is getting called. So, for the compiler, this call is Mammal.speak().

But at execution time, the JVM clearly knows that the mammal reference is holding the reference of the Cat object, so for the JVM, this call is Cat.speak()

With respect to the method it overrides, the overriding method must follow following mandatory rules:

* + It must have the same method name.
  + It must have the same arguments.
  + It must have the same return type. From Java 5 onward, the return type can also be a subclass (subclasses are a covariant type to their parents).
  + It must not have a more restrictive access modifier (if parent --> protected then child --> private is not allowed).
  + It must not throw new or broader checked exceptions.
  + A subclass can use super.overridden\_method() to call the superclass version of an overridden method.

And if both overriding methods follow the above mandatory rules, then they:

* + May have a less restrictive access modifier (if parent --> protected then child --> public is allowed).
  + May throw fewer or narrower(parent class has parent exception and child class has child exception) checked exceptions or any unchecked exception. If parent class has unchecked exception and child has checked then it throws an error.
  + When child has unchecked exception(Arithmetic) and parent has checked Exception(IOException) it compile sucessfully.  
    but for vice versa gives error
  + For checked Exception ,the different checked exception gives error only parent child applicable but for unchecked exception same level different exception allowed.

* + There is no effect on the overridden method if the method in the superclass is declared as synchronized or strictfp.

Diagram

Description automatically generated

Table

Description automatically generated

 ClassNotFoundException vs NoClassDefFoundError

Both **ClassNotFoundException**and **NoClassDefFoundError**occur when a particular class is not found at run time but under different scenarios.

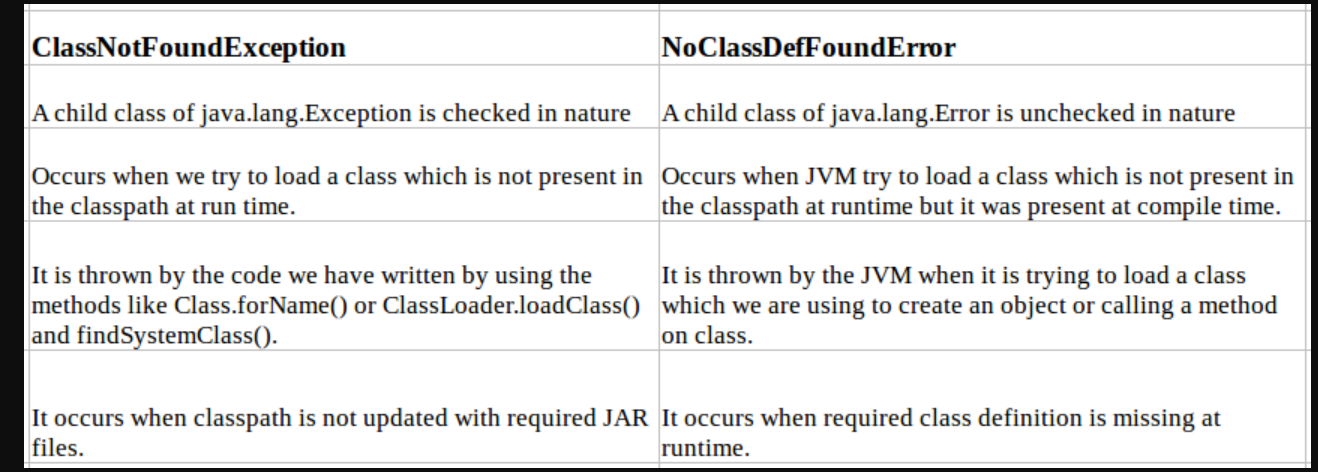
**ClassNotFoundException is a checked exception**

this exception occurs when you try to run an application without updating the classpath with required JAR files. For Example, You may have seen this exception when doing the JDBC code to connect to your database i.e.MySQL but your classpath does not have JAR for it.

**NoClassDefFoundError** Is a subtype of java.lang.Error and Error class indicates an abnormal behavior which really should not happen with an application but and application developers should not try to catch it, it is there for JVM use only

NoClassDefFoundError occurs when JVM tries to load a particular class that is the part of your code execution (as part of a normal method call or as part of creating an instance using the new keyword) and that class is not present in your classpath but was present at compile time because in order to execute your program you need to compile it and if you are trying use a class which is not present compiler will raise compilation error.

**Conclusion**



|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |
| 3) | StringBuffer was introduced in Java 1.0 | StringBuilder was introduced in Java 1.5 |

The contract between equals() and hashCode() is:

1) If two objects are equal, then they must have the same hash code.

2) If two objects have the same hash code, they may or may not be equal.

OOPS principles:

The 4 principles of object oriented programming:

1.Abstraction

2.Encapsulation

3.Inheritance

4.Polymorphism

What is oops concept?

Oops is a programming concept that works on the principles of abstraction, encapsulation,Inheritance and polymorphism. It allows user to create objects they want and create methods to handle those objects.

The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results.

Exception:

Try with resources:

In Java, the try-with-resources statement is a try statement that declares one or more resources. The resource is as an object that must be closed after finishing the program. The try-with-resources statement ensures that each resource is closed at the end of the statement execution.

*Multicatch block:*

* + At a time only one exception occurs and at a time only one catch block is executed.
  + All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Exception Chaining:

When one exception is chained to the another, it describes the cause of that exception. Constructors of the Throwable class support chained exceptions in Java. They are as follows −

Throwable(Throwable cause)  
Throwable(String msg, Throwable cause)

initCause()

getCause()

*Example*

public class Example {  
   public static void main(String[] args) {  
      try {  
         // creating an exception  
         ArithmeticException e = new ArithmeticException("Apparent cause");  
         // set the cause of an exception  
         e.initCause(new NullPointerException("Actual cause"));  
         // throwing the exception  
         throw e;  
      } catch(ArithmeticException e) {  
         // Getting the actual cause of the exception  
         System.out.println(e.getCause());  
      }  
   }  
}