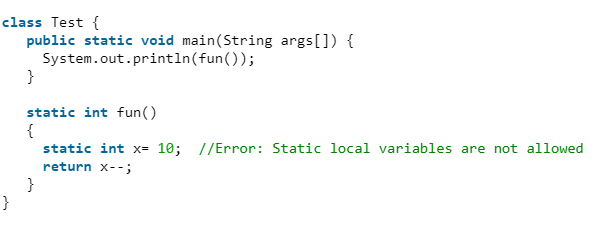
# **Static Keyword**

## ****Can a class be static in Java****

## ****Are static local variables allowed in Java?****

Static local variables are not allowed in Java. For example, following Java program fails in compilation with error “Static local variables are not allowed”.

In Java, a static variable is a class variable (for whole class). So if we have static local variable (a variable with scope limited to function), it violates the purpose of static. Hence compiler does not allow static local variable.



# OOPS Concepts

## Polymorphism (First Concept)

The word polymorphism means having many forms. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.

In Java polymorphism is mainly divided into two types:

* Compile time Polymorphism: function overloading or operator overloading.
* Runtime Polymorphism: It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime.

## Dynamic Method Dispatch or Runtime Polymorphism in Java

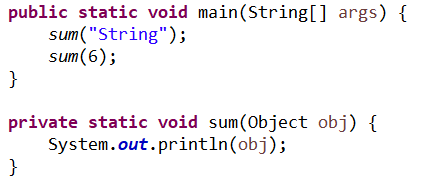
## Overloading in Java

Overloading allows different methods to have the same name, but different signatures (arguments) where the signature can differ by the number of input parameters or type of input parameters or both. Overloading is related to compile-time (or static) polymorphism.

## **What if the exact prototype does not match with arguments?**

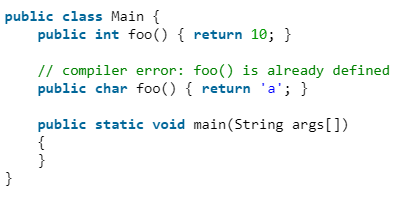
1. Type Conversion but to higher type (in terms of range) in same family.
2. Type conversion to next higher family (suppose if there is no long data type available for an int data type, then it will search for the float data type).

Example is if we have passed a String as an argument, it will search for String first if not available then will go to step higher in the data type family. If say Object is present, it will go to object method



## **Can we overload methods on return type?**

We **cannot** overload by return type.



Can we overload static methods?  
The answer is ‘Yes’. We can have two or more static methods with same name, but differences in input parameters.

Can we overload methods that differ only by static keyword?  
We **cannot** overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same).

## **Can we overload main () in Java?**

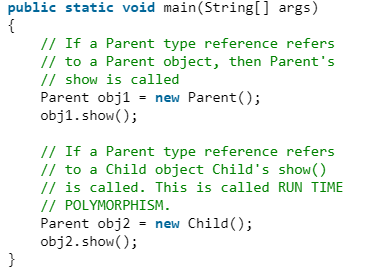
Like other methods we can overload the main method too as JVM also looks for signature of method.

## **What is the difference between Overloading and**[Overriding](https://www.geeksforgeeks.org/overriding-in-java/)**?**

Overloading is about same function have different signatures. Overriding is about same function, same signature but different classes connected through inheritance.

## Overriding in Java

In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature, and same return type (or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.



## **Overriding and Access-Modifiers**

The [access modifier](https://www.geeksforgeeks.org/access-modifiers-java/) for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the super-class can be made public, but not private, in the subclass. Doing so, will generate compile-time error.

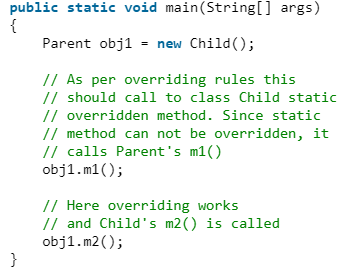
## **Final methods cannot be overridden**

## If we don’t want a method to be overridden, we declare it as [final](https://www.geeksforgeeks.org/final-keyword-java/).

## **Static methods cannot be overridden (Method Overriding vs. Method Hiding)**

When you define a static method with same signature as a static method in base class, it is known as [method hiding](https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/). Static methods can be called using the class name itself. Parent.methodName () will to parent class method and Child.methodName () will go to child class method. But still we want to use the instance, then using the parent class reference it will go to the parent class only even if instance is of child class.

Parent parent = new Child (); <= It will go to parent class even because the static method in child class is hidden to parent class and it is called method hiding too.



## **Private methods cannot be overridden**

Clear

## **The overriding method must have same return type (or subtype)**

Clear

## Overriding and constructor

We cannot override constructor as parent and child class can never have constructor with same name (Constructor name must always be same as Class name).

## **Overriding and Exception-Handling**

**Rule#1:** If the super-class overridden method does not throw an exception, subclass overriding method can only throws the [unchecked (Runtime) exception](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/), throwing checked exception will lead to compile-time error.

**Rule#2:** If the super-class overridden method does throws an exception, subclass overriding method can only throw same, subclass exception. Throwing parent exception in [Exception hierarchy](https://www.geeksforgeeks.org/exceptions-in-java/) will lead to compile time error. Also there is no issue if subclass overridden method is not throwing any exception.

## ****Overriding and abstract method****

Abstract methods in an interface or abstract class are meant to be overridden in derived concrete classes otherwise a compile-time error will be thrown.

## **Overriding and synchronized/strictfp method**

The presence of synchronized/strictfp modifier with method have no effect on the rules of overriding, i.e. it’s possible that a synchronized/strictfp method can override a non-synchronized/strictfp one and vice-versa.

# Exception

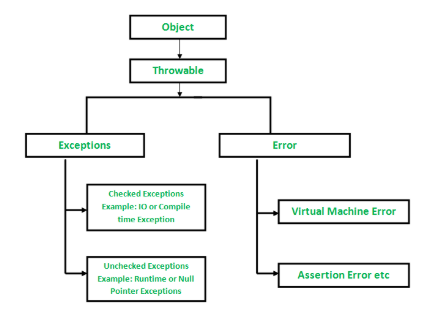
An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time that disrupts the normal flow of the program’s instructions. An Exception “indicates conditions that a reasonable application might want to catch.” Exceptions are the conditions that occur at runtime and may cause the termination of program. But they are recoverable using try, catch and throw keywords. Exceptions are divided into two categories: [checked and unchecked exceptions](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/).

## **Error**

An Error indicates serious problem that a reasonable application should not try to catch. Errors are the conditions which cannot get recovered by any handling techniques. It surely cause termination of the program abnormally. Errors belong to unchecked type and mostly occur at runtime. Some of the examples of errors are Out of memory error or a System crash error.

E.g. java.lang.StackOverflowError

## Exception Hierarchy



## Checked Exceptions

Checked Exceptions are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws*keyword. E.g. IOException.

## Unchecked Exceptions

Unchecked are the exceptions that are not checked at compiled time. E.g. ArrayIndexOutOfBoundException.

## Inheritance (Concept 2)

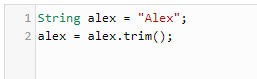
<https://www.geeksforgeeks.org/inheritance-in-java/>

# Serialization

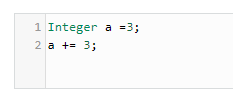
# Immutable Classes

Immutable class means that once an object is created, we cannot change its content. In Java, all the [wrapper classes](https://www.geeksforgeeks.org/wrapper-classes-java/) (like Integer, Boolean, Byte, Short) and String class is immutable. We can create our own immutable class as well. An object is immutable if its state cannot change after construction. Immutable objects don’t expose any way for other objects to modify their state; the object’s fields are initialized only once inside the constructor and never change again.

**String**is the most popular immutable class in Java. Once initialized its value cannot be modified. Operations like *trim(), substring(), replace()* always return a new instance and don’t affect the current instance.



Another example from JDK is the wrapper classes like: *Integer, Float, Boolean* … these classes don’t modify their state, however they create a new instance each time you try to modify them.



## How to create immutable class

<https://dzone.com/articles/how-to-create-an-immutable-class-in-java>