Method Overloading vs. Method Overriding

I. The conditions for method overloading

1.) The ***number*** of parameters is different for the methods.
2.) The parameter ***types*** are different (such as changing a parameter that was a float to an int).

II. How to NOT overload methods:

It's also very important to understand that method overloading is NOT something that can be accomplished with either, or both, of these two things:

1. DO NOT just change the return type of the method.

If the return type of the method is the only thing changed, then this will result in a compiler error.

2. DO NOT just change the name of the method parameters, without changing the parameter types.

If the name of the method parameter is the only thing changed then this will also result in a compiler error.

Examples of *incorrect* Method Overloading in Java

```
int changeDate(int Year) ;
float changeDate (int Year);
```

compiler error - can't overload based on the type returned (one method returns int, the other returns a float):

```
int changeDate(int Year);
int changeDate(int Month) ;
```

compiler error - can't overload by changing just the name of the parameter (from Year to Month):

Correct method overloading

Valid case of overloading, since the methods have different number of parameters:

```
int changeDate(int Year, int Month) ;
int changeDate(int Year);
```

Also a valid case of overloading, since the parameters are of different types:

```
int changeDate(String Year) ;
int changeDate(int Year);
```

Method overriding

Overriding methods is completely different from overloading methods. If a derived class requires a different definition for an inherited method, then that method can be redefined in the derived class. This would be considered overriding.

An overridden method would have the exact same method name, return type, number of parameters, and types of parameters as the method in the parent class, and the only difference would be the definition of the method.

Example of method overriding

```
public class Parent {

 public int anyMethod() {

   return 3;

    }
}
```

```
public class Child extends Parent{

 // this is method overriding:
 public int anyMethod() {

    return 4;

    }

}
```

In the example anyMethod is an overridden method in the Child class, because it has **the exact same name, number of parameters, and return type** as the anyMethod defined inside it's parent class (named Parent).

Overriding happens at run time, Overloading at compile time.

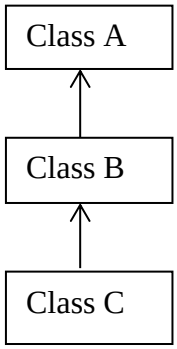
Summary of differences between overloading and overriding

When overloading, one must change either the type or the number of parameters for a method that belongs to the same class.

Overriding means that a method inherited from a parent class will be changed.

But, when *overriding* a method everything remains exactly the same except the method definition – *basically what the method does is changed slightly to fit in with the needs of the child class.* But, the method name, the number and types of parameters, and the return type will all remain the same.

Method overriding is a run-time phenomenon that is the driving force behind polymorphism.



Example: If class A is a parent of class B, class C extends B and all classes implement the instance method void doSomething(). Is it possible for the version of the doSomething() method in class A to be called by an instance of class B?

It is **not** possible with an instance of B. Because B overrides the doSomething() method, calling the A class's version of the doSomething() method is not possible with an ***instance*** of B.

However, this is possible within the definition of one of the methods of B using a call to super. The syntax for this would be ***super.doSomething()***