**Data Types:**

**char** - The **char**type is used to hold a single character. It is created like Strings; however, it uses single quotes for the value.  
**int**- The **int**type is used to store whole numbers (or **integers**, as we call them in programming).  
**boolean** - It can hold only the values **true**or **false**. This is handy when we work with states or conditions.

**Difference between float and double:**

**float** – It is memory saved us, because here we are using 7 decimal values, and it has 4 bytes 32 bits. float: Use when memory is limited, and precision up to 7 decimal places is sufficient (e.g., games, graphical applications).  
**double** – Compared to float double is more memory required, because we are using 15-16 decimal values here, and it has 8 bytes 64bits. double: Use when higher precision is required, such as scientific, financial, or engineering applications.

* In Java, the default data type for decimal numbers is double.
* Example: Writing 3.14 is treated as a double unless explicitly specified as a float (e.g., 3.14f).

**Packages**:

1. Packages in java are folders created to store programs in organized manner.
2. A **java package** is a group of similar types of classes, interfaces and sub-packages.

**Math Operator:**

Math is fun! Here are some key takeaways: You can use basic math operators to perform calculations with values and variables.

+ is addition

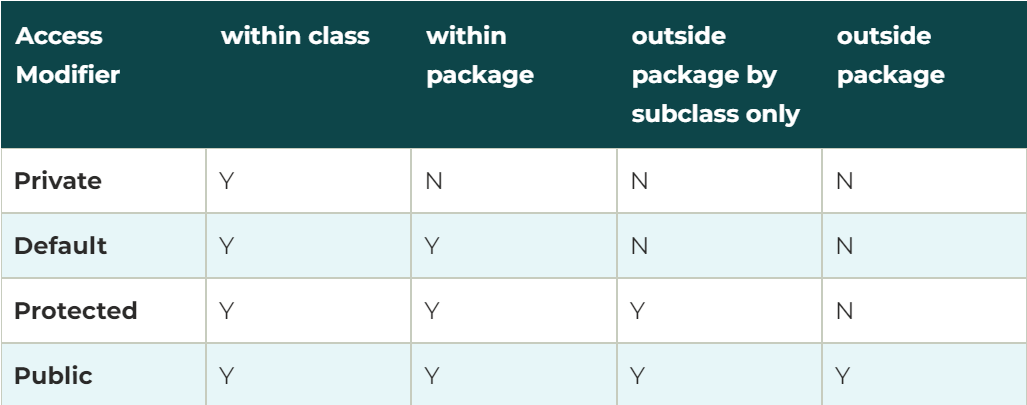
- is subtraction

/ Is division

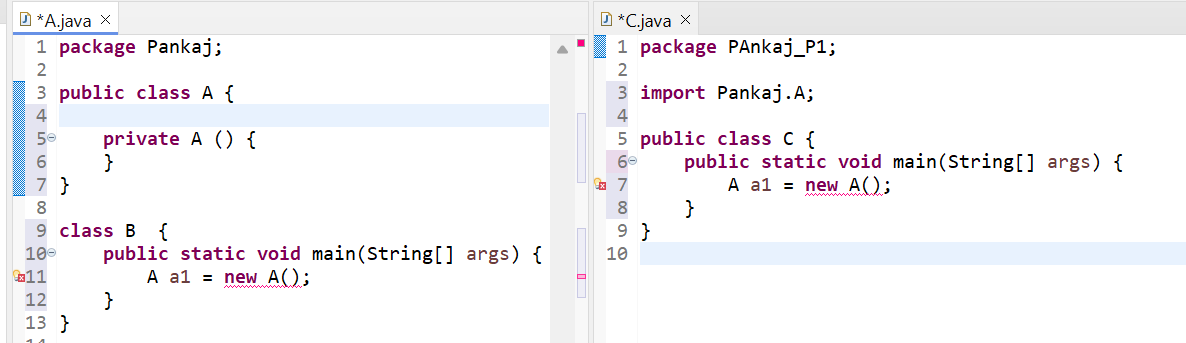
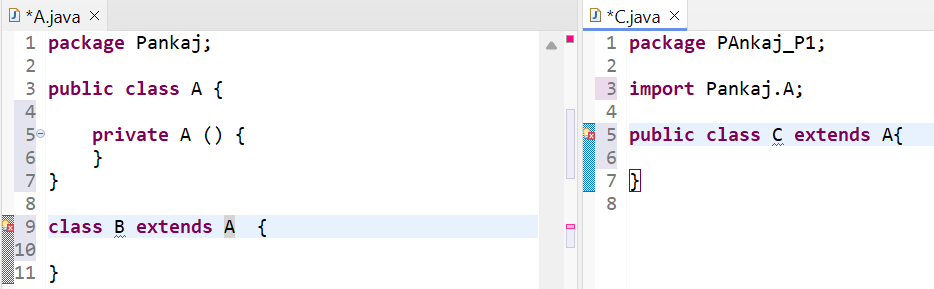
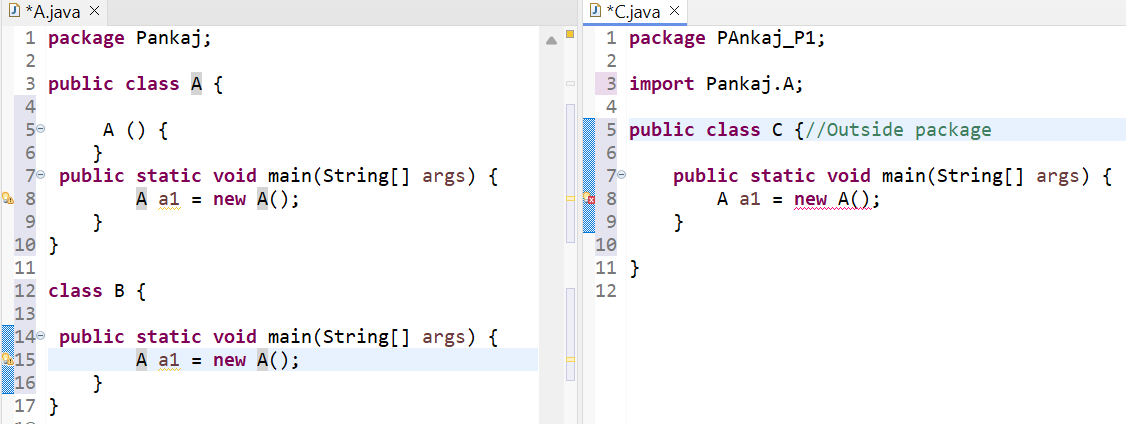
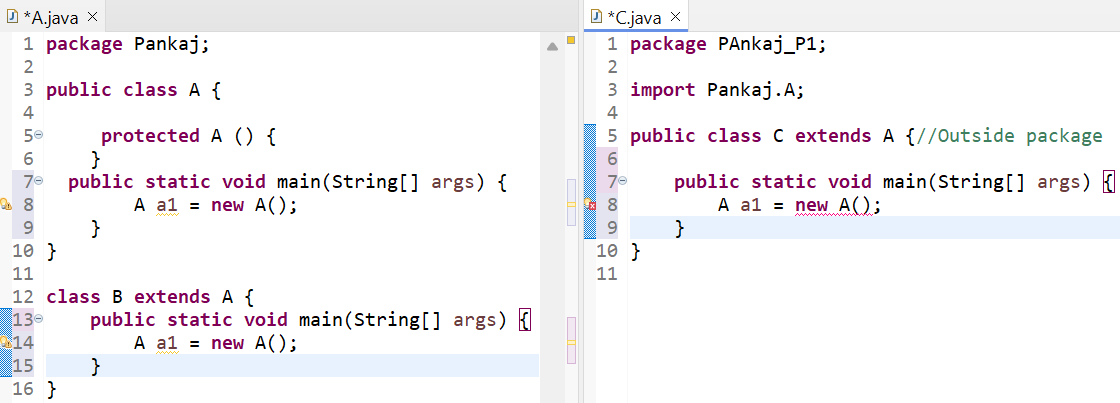
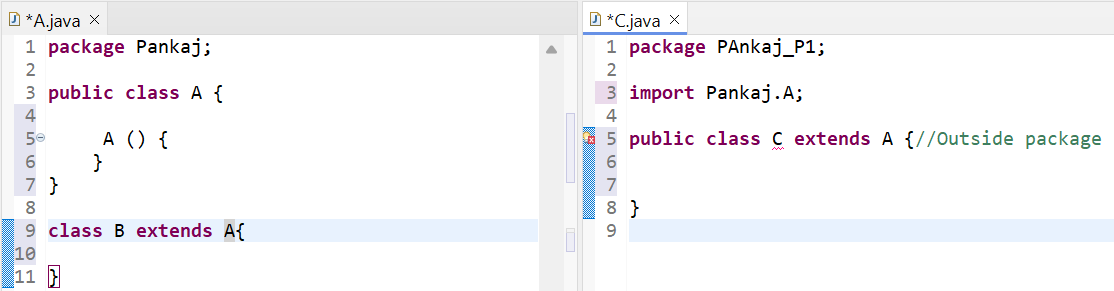
% finds the remainder of a division.

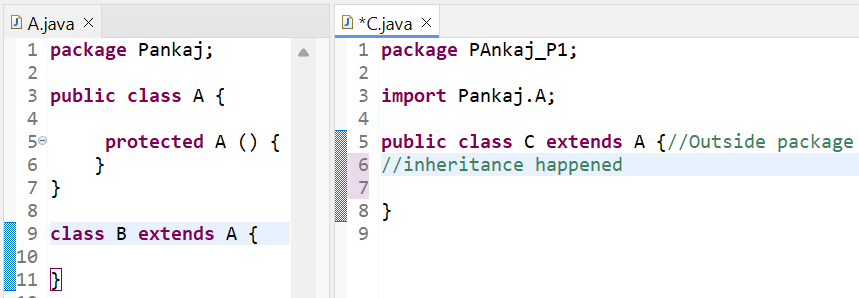
We can add Strings using the + operator, in a process called concatenation.

**Access modifiers**:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If we do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If we do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

**Access Specifier Constructor perspective**:

1. If we make a constructor **private** then it’s object cannot be created outside the class but inside the class, we can be able to create.
2. ****If we make a constructor **private** inheritance is not allowed.
3. If we make a constructor as **default**, then it’s object can be created in same package but not outside package.
4. If we make a constructor **default**, then inheritance can be done within the package only.
5. ****If we make a constructor **protected**, it’s object can be created in same package.
6. If we make a constructor **protected**, then inheritance can be done within the package and different package also.



**What is data hiding?**

Here we make a variable as private, so that variable cannot accessed to the outside the class. This is called as data hiding.

**Interface in java?**

* An interface can consist of only incomplete methods in it.  
  **Note**: INCOMPLETE METHODS ARE ALSO CALLED AS ABSTRACT METHODS.

Ex: public interface A { public interface A {  
 public void test () { public void test ();  
 } //error }//No error

**Polymorphism in java?**

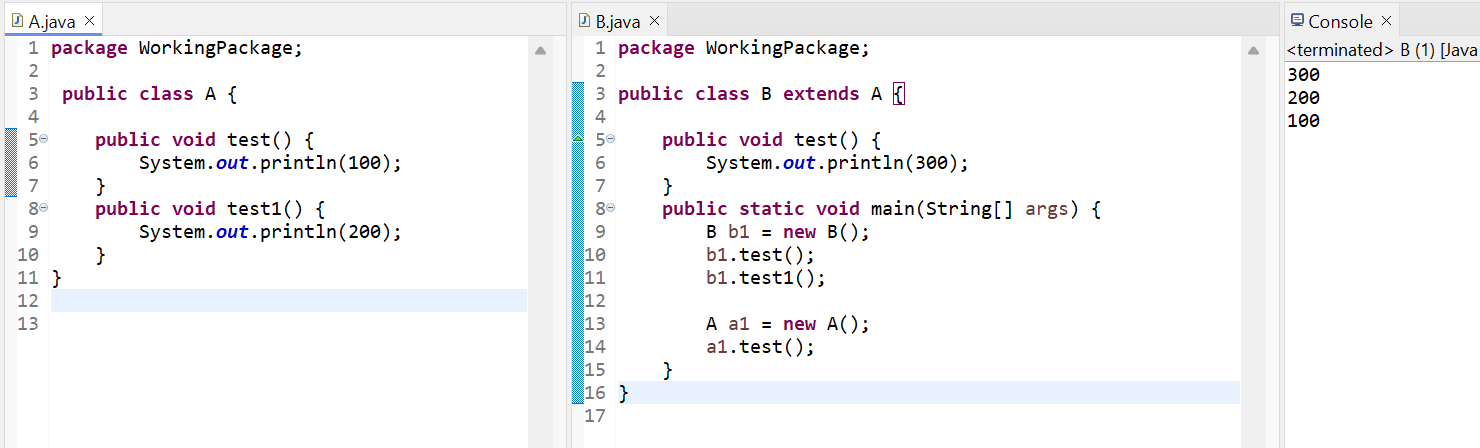
* Here we develop a feature such that it can take more than one form.

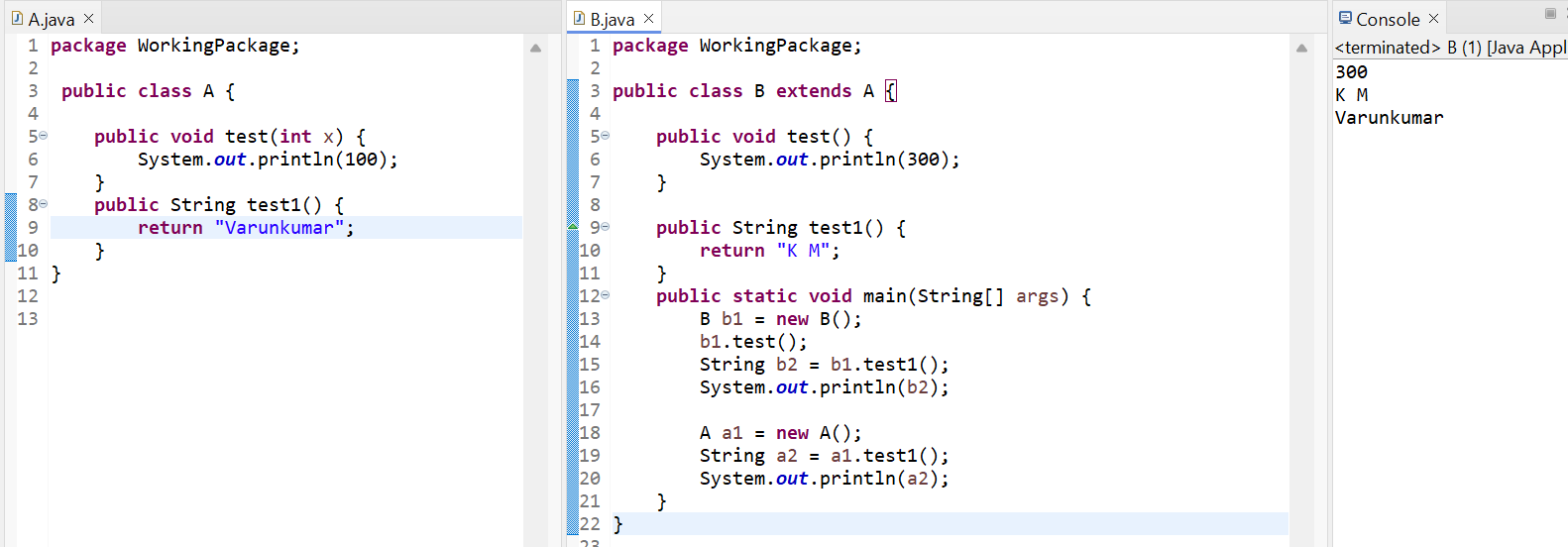
**Note**: Polymorphism is applicable only on methods. It’s not applicable for variables.

There are two types of polymorphisms:

1. **Method Overriding**

* For overriding inheritance is mandatory
* Here we are developing a method from one class to another class with the intension of reusing it or modifying it. So, this is called overriding.
* Why it is polymorphism here two methods with the same name.



****

In java version 5, introduced annotations.

**@Overide** – Override annotation is help us to check weather Overriding is happening or not we use @Override annotation. This annotation will instruct the compiler to check weather overriding is happening or not.

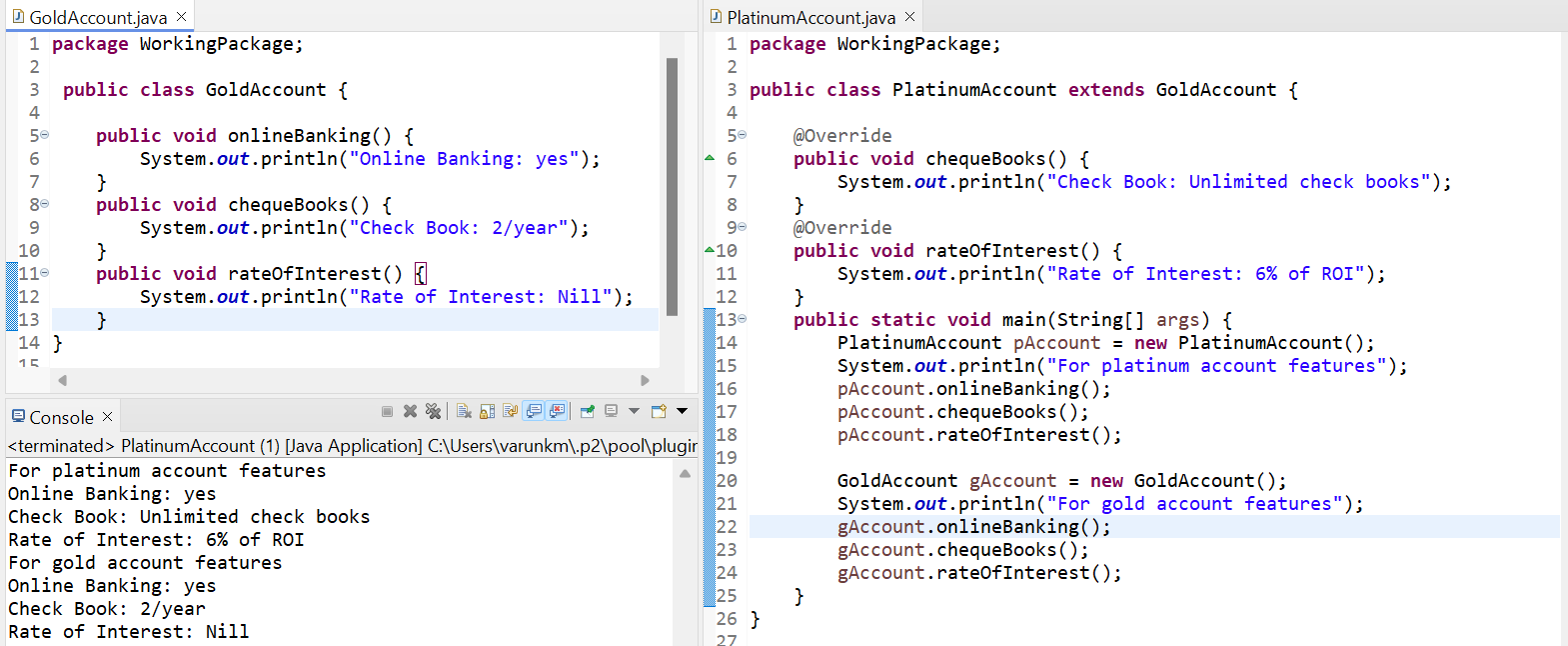
A screenshot of a computer

Description automatically generated

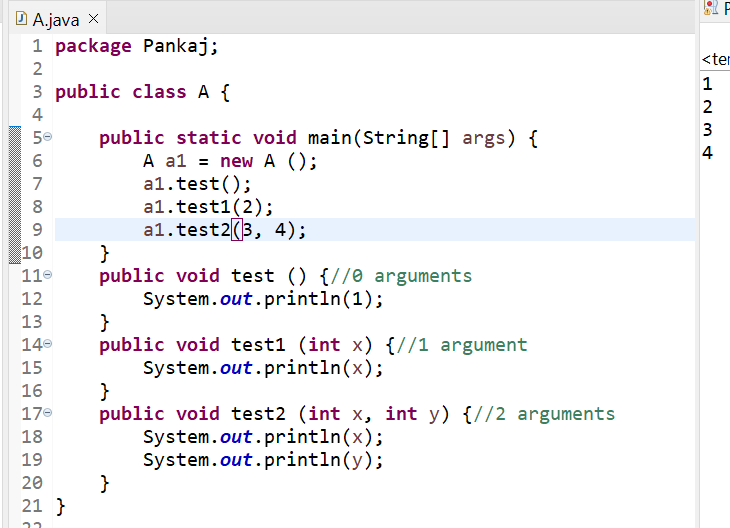
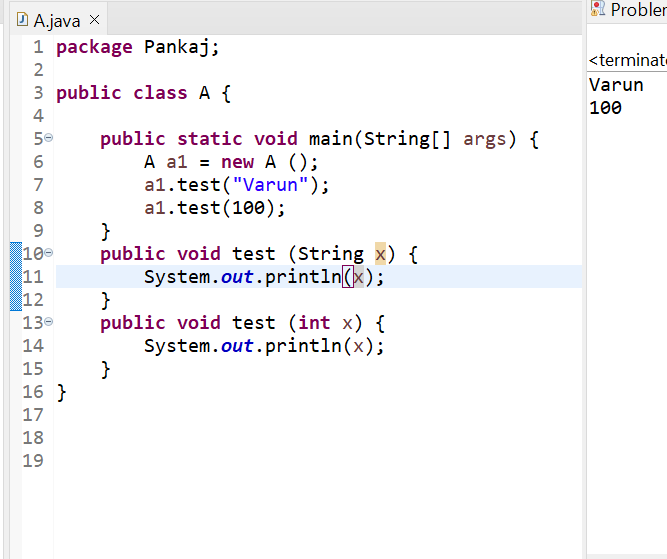
**Scenario example of overriding method** - **HDFC Bank**:

Gold Account Platinum Account

* Online Banking Online Banking
* Cheque Book 2 per Year Unlimited Cheque Books
* Nill ROI 6% of ROI (rate of interest)

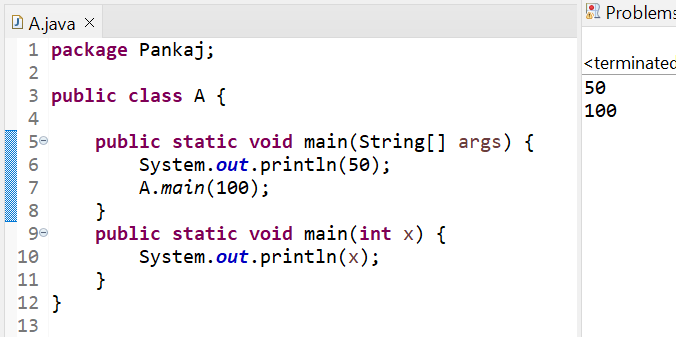


1. **Method Overloading**

Here we create a more than one method in same class and same name provided with different number of arguments and different types of arguments.

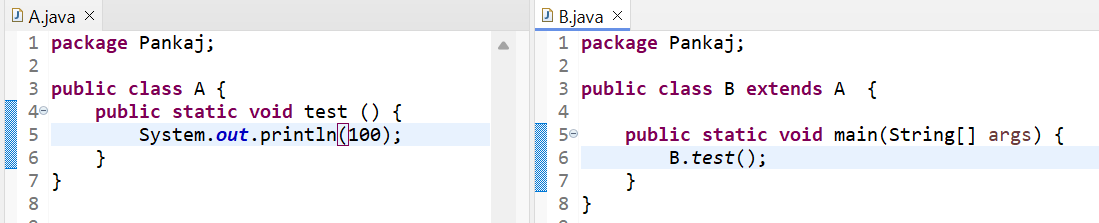
**Can we create a more than one method?**

* More than one method can we create, but the number of arguments in that method should be different.

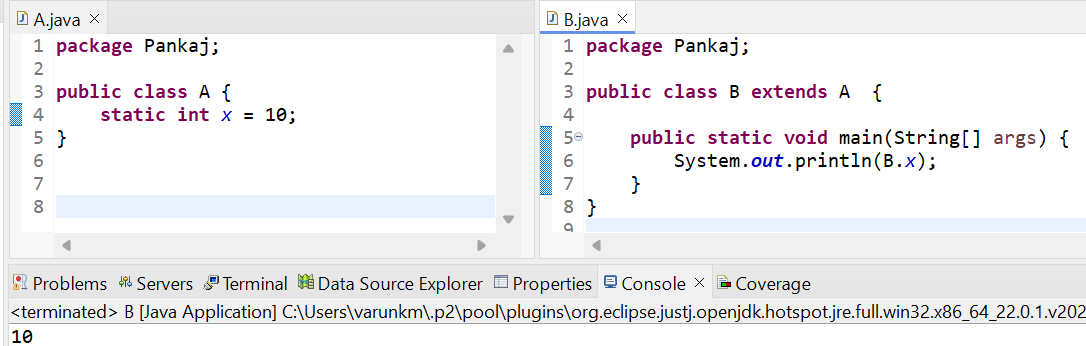


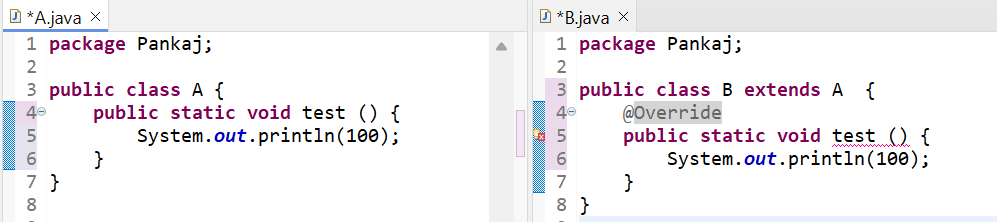
**Can we override static method?**

* When we call child class name parent member that will work. But inheritance is not happening.
* Inheritance of static method and variable is doesn’t happen.
* Any Opps concept static never happen.



When we run this program B.test () will convert into A.test () because of this internal conversion this method gets called and hence this program runs.





**Overriding with access specifier:**

Whenever we are doing overriding the method that will override the inherited methods scope should be same or higher, but not lower.  
*If we are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.*

**Access modifier restrictions in decreasing order:**

* Private
* Default
* Protected
* Public