Last updated: 13 Oct 2021

Selenium Java Training - Session 9 - Java (Part 7) - Inheritance, Overriding and Modifiers

Java (Part 7) - Inheritance, Overriding and Modifiers

Inheritance

Inheritance is a mechanism in which one class acquires the properties (i.e. variables and methods) of another class

- The purpose of this Inheritance is to use the properties (i.e. methods and variables) inside a class instead of recreating the same properties again in new class.
- Child class acquires the properties (i.e. variables and methods) of Parent Class.
- Child class uses **extends** keyword to inherit the properties from parent class
- Demonstrate a child class which inherits the properties from Parent Class - Demonstrate here
    - Child class can have specific properties (i.e. variables and methods) which are not available in the parent class
    - Object created for parent class can access the variables and methods that are created in parent class only. It cannot access the child class properties.
    - Object created for child class which is inheriting the parent class can access the variables and methods of both parent class and child class.

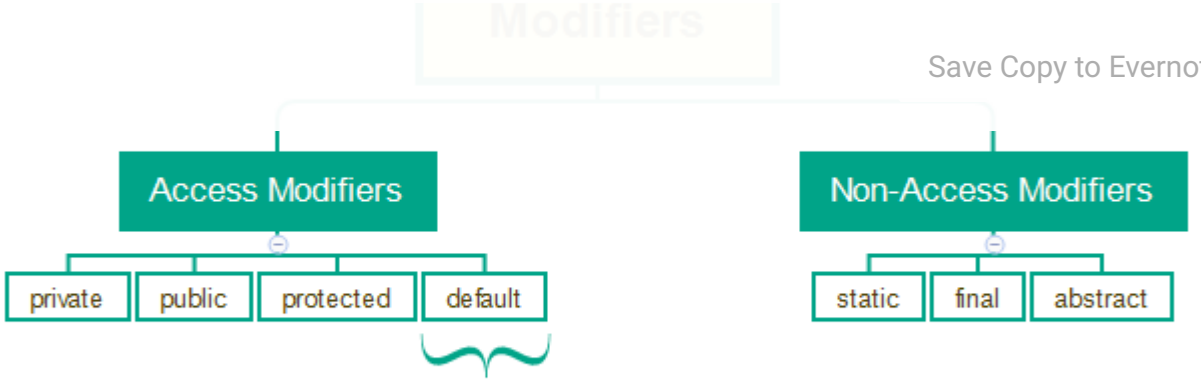
Overriding

When a method in the Child class (i.e. sub-class) is duplicate of a method in Parent class (i.e. super-class) , then the method in the sub-class is said to override the method in super-class.

- When we create an Object for Sub-class and call the overridden method, the method in the sub-class will be called - Demonstrate here
- Even though the name of the method in the sub-class has the same name as a method in super-class, if the type of parameters or number of parameters, then the method in the sub-class will overload the method in super-class instead of overriding
- Constructors cannot be overridden as the name of the constructor needs to be same as the name of the Class.

Modifiers

- Modifiers in Java can be categorized as below:



When we don't specify any modifier

public Access Modifier

- Classes/variables/methods specified with 'public' access modifier can be accessed directly by the classes which are in the same package - Demonstrate
- Classes/variables/methods specified with 'public' access modifier can be accessed by the classes outside the package after importing the classes - Demonstrate

private Access Modifier

- Java classes cannot be specified with 'private' access modifier - Demonstrate
- variables/methods specified with 'private' access modifier can be accessed only with in the same class - Demonstrate

default Access Modifier

- When no modified is specified before classes/variables/methods, then we name it as default modifier - Demonstrate
- default means public to all the classes inside the same package and private to the classes which are outside the package - Demonstrate

protected Access Modifier

- protected means public to all the classes inside the same package and private to all the classes which are outside the package except child classes
- Java classes cannot be specified with 'protected' access modifier - Demonstrate
- While accessing the protected variables/methods outside the packages using sub-classes, we don't have to create an object to access them as they are inherited variables and methods - Demonstrate

static Non-Access Modifier

- Java classes cannot be specified with 'static' non-access modifier - Demonstrate
- Variables declared directly inside the class but outside the methods and are specified with 'static' modifier are known as static variables

- Memory allocated to the static variables is different from the memory allocated to the instance variables - View here
- static variables needs to be accessed with the help of Class name, as they belong to the Class memory - View here
- static variables are generally used to store common data, where as Object variables/Instance variables are used to store Object specific data.
    - wheels variable can be used as a static variable/class level variable as it has common data i.e. wheels count is 4 for all the cars in the market
    - Where as cost variable cannot be used as a static variable as its value changes from car to car, hence we use it as an Object variable/Instance variable.
- static can also be used with methods
- static can only access static stuff
    - You have to create object to overcome this

final Non-Access Modifier

- The value of the variable cannot be changed on specifying it with final non-access modifier - Demonstrate here
- final modifier specified classes cannot be inherited/extended by other classes - Demonstrate here
- final modifier specified methods in a class cannot be overridden by its sub-classes - Demonstrate here

abstract Non-Access Modifier

- variables cannot be specified with 'abstract' non-access modifier - Demonstrate
- On specifying a method with abstract modifier, we can just declare the method without implementing it - Demonstrate here
- Classes having at-least one abstract specified method must be specified as abstract
- Sub-Class inheriting the Super-Class needs to implement the abstract specified methods in Super-Class - Demonstrate here
    - Purpose of abstract methods - Used when the super-class dont have to implement everything, and when the sub-classes inheriting the super-class needs to implement them.
- Objects cant be created for abstract classes, we have to create a Sub-Class and access its variables/methods using Sub-Class object reference - Demonstrate here

By,
Arun Motoori