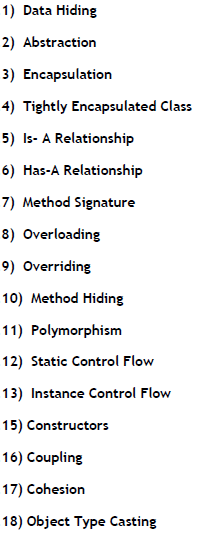
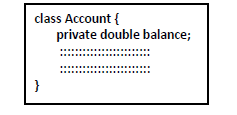
# OOPS



**Data Hiding:**

* **Our Internal Data should Not go out Directly OR Outside Person should not Access Our Internal Data Directly. This is the Concept of Data Hiding.**
* **Eg: After Providing Proper User Name and Password Only we can able to Access Our Mail Information.**
* **Eg: After swiping ATM Card and Providing Valid Pin Number we can able to Access Our Account Information. By declaring Data Member as private we can achieve Data Hiding.**



**The Main Advantage of Data Hiding is Security.**

**Note: Recommended Modifier for Data Members is private.**

**Abstraction:**

* **Hiding Internal Implementation and Highlight the Set of Services which are offering is the Concept of Abstraction.**
* **Eg: By using Bank ATM GUI Screen Bank People are highlighting the Set of Services what they offering without highlighting Internal Implementation.**

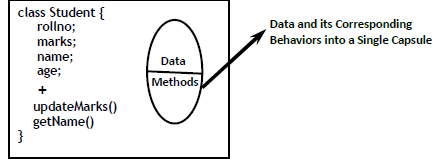
**The Main Advantages of Abstraction are:**



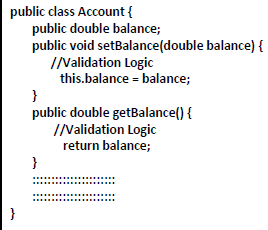
* **We can Achieve Security as we are not highlighting Our Internal Implementation.**
* **Without effecting Outside Person we can able to Perform any Type of Changes in Our Internal Design. Hence Enhancement will become Easy.**
* **It Improves Maintainability and Modularity of the Application.**

**Encapsulation:**

**The Process of Binding Data and Corresponding Methods into a Single Unit is Called Encapsulation.**



**In any Component that follows Data Hiding and Abstraction, Such Type of Component is Called Encapsulated Component.**





**The Main Advantages of Encapsulation are**

* **We can Achieve Security.**
* **Enhancement will become Very Easy.**
* **It Improves Maintainability and Modularity of the Application.**

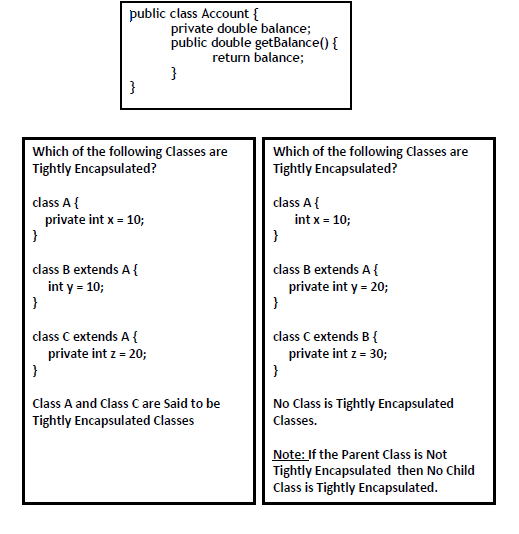
**The Main Advantage of Encapsulation is we can Achieve Security and the Main Disadvantage of Encapsulation is it Increases Length of the Code and Slow Down Execution. So that Performance will be Impacted.**

**Tightly Encapsulated Class:**



* **A Class is Said to be Tightly Encapsulated if and only if Each and Every Variable of that Class declared as private.**

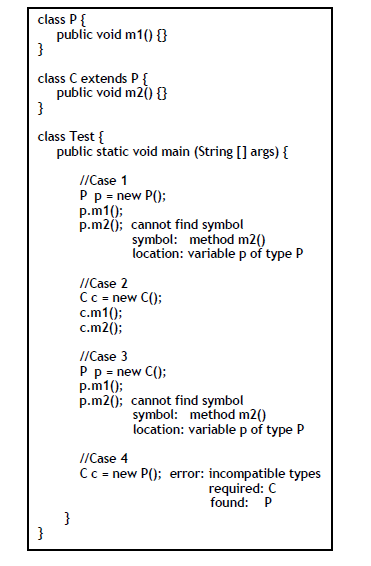




**IS-A Relationship (Inheritance):**



* **By using extends KeyWord we can implement IS-A Relationship.**
* **The Main Advantage of Inheritance is Re-usability of the Code.**



**Conclusions:**

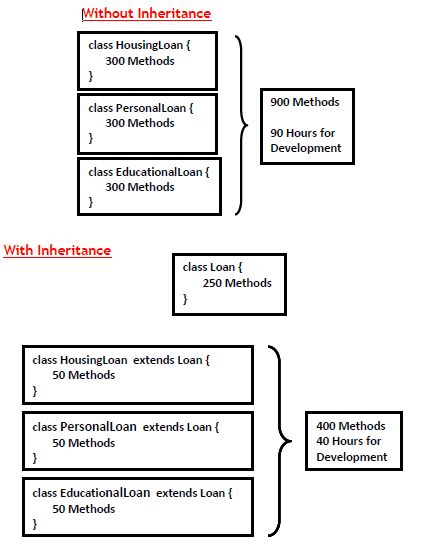
1. **Whatever Methods Parent has, by Default Available to the Child. Hence on Child Class Object we can Call Both Parent and Child Class Methods.**

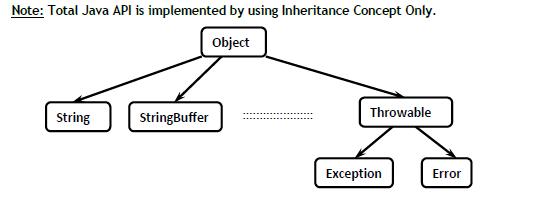
**2) Whatever Methods Child has by Default Not Available to the Parent and Hence on the Parent Class Reference we can't Call Child Specific Methods.**

**3) Parent Reference can be used to hold Child Object. But by using that Reference we can Call Only Methods Available in Parent Class and we can't Call Child Specific Methods.**

**4) Child Reference cannot be used to hold Parent Object. But Parent Reference can be used to hold Child Object.**

1. **Parent Class contains the Common Functionality which required for Child Class. Whereas Child Class contains Specific Functionality.**





* **The Most Common Methods which are required for All Java Classes are defined Inside Object Class. Hence Object Class Acts as Root for All Java Classes.**
* **The Most Common Methods which are required for All Exceptions and Errors are defined in Throwable Class. Hence Throwable Class Acts as Root for Exception Hierarchy.**

**Multiple Inheritance:**

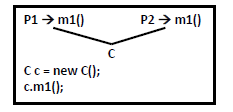
**A Java Class can't extend More than One Class at-a-Time. Hence Java won't Provide Support for Multiple Inheritance with Respect to Classes.**

**Eg: class A extends B, C { } ✖**

**Why Java won't Provide Support for Multiple Inheritance?**



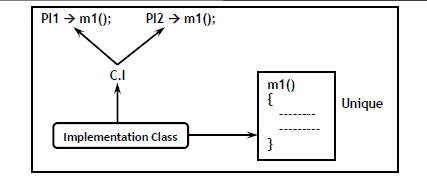
**There May be a Chance of Raising Ambiguity Problems.**

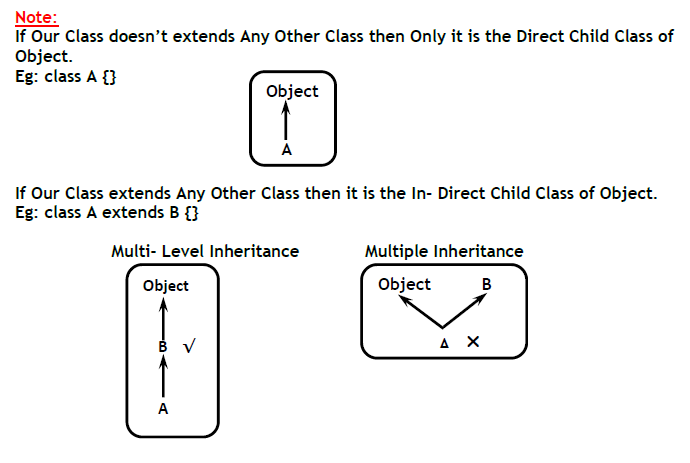


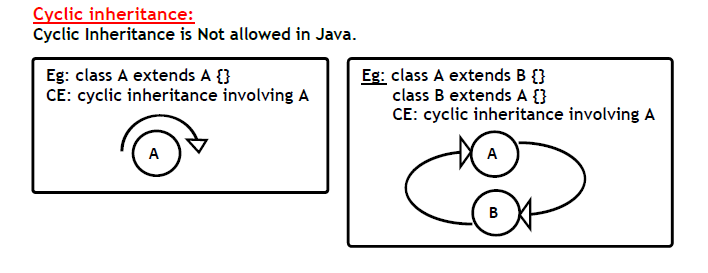
* **An Interface can extend any Number of Interfaces at a Time. Hence Java Provides Support for Multiple Inheritance with Respect to Interfaces.**



**Even though Multiple Method Declarations Present, but Implementation is Unique. Hence there is No Chance of Ambiguity Problem in Interfaces.**

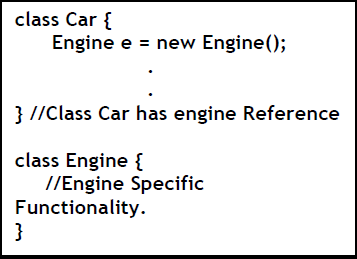






**HAS-A Relationship:**

* **It is Also Known as *Composition* OR *Aggregation.***
* **There is No Specific Keyword to implement HAS-A Relationship. But Most of the Times we are depending on *new* Keyword.**
* **The Main Advantage of HAS-A Relationship is Code Re- Usability.**

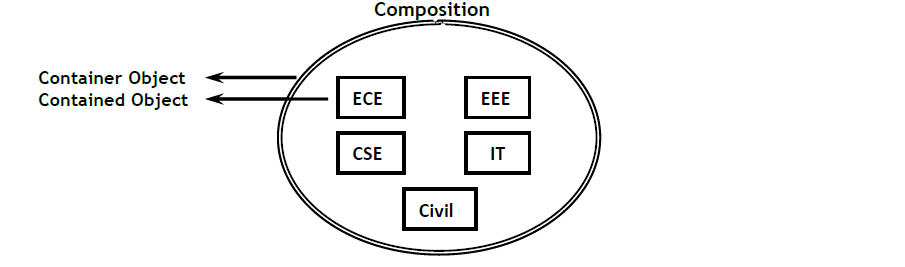


**Composition Vs Aggregation**

**Composition:**

**Without Existence of Container Object, if there is No Chance of Existence of Contained Objects then *Container* and *Contained* Objects are Said to be Strongly Associated and this *Strong Association* is known as Composition.**

**Eg: A University has Several Departments. Without Existence of University there is No Chance for Existence of Departments Objects. Hence *University* and *Departments* are Strongly Associated and this Strong Association is Known as *Composition.***

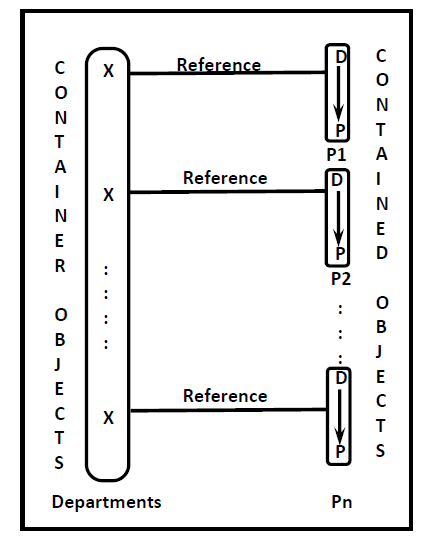


**Aggregation**

**Without Existence of Container Object, if there is a Chance of Existence of Contained Objects then *Container* and *Contained* Objects are Said to be**

**Weakly Associated and this *Loose Association* is known as Aggregation.**

**Eg: Within a Department there May be a Chance of working Several Professors. Without Existence of Departments Object there May be a Chance of existing Professors Objects. Hence Department and Professors are Loosely Associated and this *Loose Association* is Known as Aggregation.**



**Note:**

* **In Composition Objects are Strongly Associated whereas in Aggregation Objects are Weakly Associated.**
* **In Composition Container Object holds Contained Objects Directly whereas in Aggregation Container Object Just Holds References of Contained Objects.**

**Overloading :**

* **2 Methods are Said to be Overloaded if and only if Both Methods are having Same Name but Different Types of Arguments.**
* **In C Language Overloading Concept is Not there hence we can't Declare 2 Methods with the Same Name but different Type of Arguments. Hence If there is a Change in Argument Type then Compulsory we should go for New Method Name.**

**Eg: abs(int)**  **abs (10);**

**labs(long)**  **labs (10l);**

**fabs(float)**  **fabs (10.5f);**

* **Lack of Overloading in C Increases the Complexity of the Programming.**
* **But in Java we can Declare Multiple Methods with the Same Name but with different Arguments Types and these Methods are Called Overloaded Methods.**

**Eg: abs(int),**

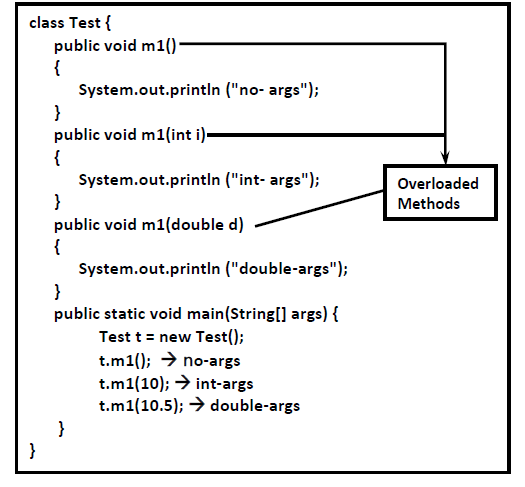
**abs(long),**

**abs(float)**

**Having Overloading Concept in Java Reduces the Complexity of the Programming.**

**Case 1:**

**The Overloading Method Resolution is the Responsibility of Compiler based on Reference Type and Method Arguments. Hence Overloading is Considered as *Compile-Time Polymorphism* OR *Early Binding.***



**Case 2: Automatic Promotion in Overloading**

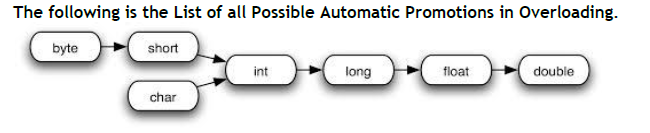
* **While Resolving Overloaded Methods if Exact Method with the required Argument is Not Available then the Compiler won’t Raise Immediately Compile Time Error.**

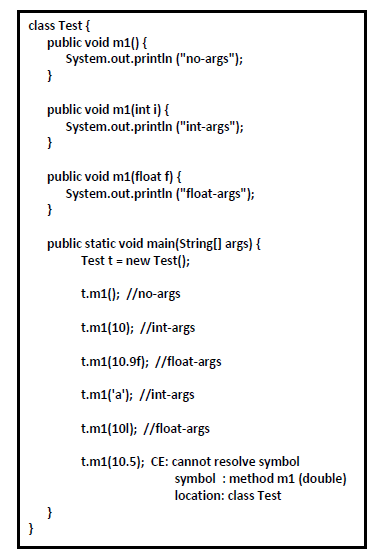


* **First Compiler will Promote Arguments to Next Level and Check is there any Matched Method with Promoted Arguments.**
* **If the Matched Method is Found then it will Considered Otherwise Compiler Promotes the Argument to the Next Level.**

**This Process will be continued until all Possible Promotions.**

**After all Possible Promotions if still the Compiler is Unable to find the Matched Method then it raises Compile Time Error.**





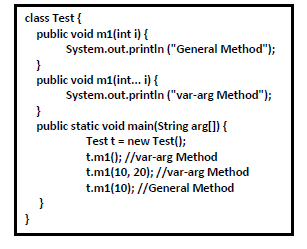
**Case 3:**

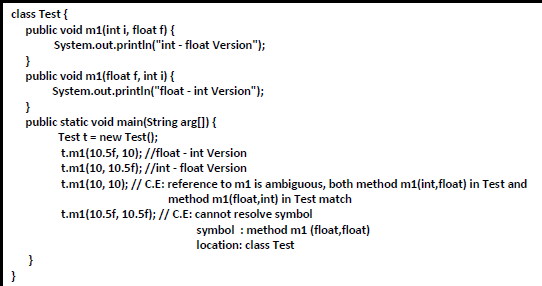


* **In Overloading exact Match will get High Priority.**
* **In Overloading Child Class Argument will get More Priority than Parent Class Argument.**

**Case 4:**

* **In General var-arg Method will get Least Priority i.e., if No Other Method Matched then Only var-arg Method will get the Chance. It is Exactly Same as *default* Case Inside *switch.***





**Case 5:**

* **Overloading Method Resolution will always take Care by Compiler based on the Reference Type but not based on Runtime Object.**

