Inheritance

04 December 2021 15:30

Package: It is a set of related reference types organized together for
(1) avoiding collisions between their name and the names of other types not belonging to this set.
(2) hiding these types and their members from other types not belonging to their set.

A type **T** that belongs to a package **p** has following characteristics
(1) it is identified by its fully qualified name which is **p.T**
(2) its binary representation is loaded by default from path **p/T.class**
(3) it is visible outside of **p** only if it is declared with public modifier (in **T.java**)

Visibility of a member outside of its defining type

Access Level	**Inside of current package**	**Outside of current package**
private	none	none
<default>	all	none
protected	all	subtypes
public	all	all

Object Identity - two objects are considered to be identical if they refer to the same instance in the memory. In Java whether an object **x** is identical to object **y** can be determined from expression **x == y**.

Object Equality - two objects are considered to be equal if they refer to two instances with matching type and state in the memory. In Java whether an object **x** is equal to object **y** can be determined from expression **x.hashCode() == y.hashCode() && x.equals(y)**.

Abstract Class	**Interface**
It is a class type which does not support instantiation but can define instance fields	It is a reference type which does not support instantiation and cannot define instance fields
It can define unimplemented instance methods which must be declared with *abstract* modifier	Its instance methods are implicitly abstract and as such its implemented instance methods must be declared with *default* modifier
It can define a constructor which can be called from sub-classes	It cannot define a constructor
It can define final as well as non-final static fields	It can only define static final fields
It can include public and non-public members	It can only include public members

It can extend exactly one class which may or may not be abstract	It can extend multiple other interfaces
A class can inherit from a single abstract class and it must override all abstract methods of that class otherwise it must be declared abstract	A class can inherit from multiple interfaces and it must implement all of their abstract methods otherwise it must be declared abstract
It is generally defined to specify the common type of state (fields) inherited by different types of objects	It is generally defined to specify the common type of behavior (methods) inherited by different types of objects

Multiple Inheritance in Java - The type-system of Java does not allow a class to inherit from multiple classes because an instance of such a class will require a layout with multiple sub-objects out of which only first one can be referenced in a safe manner and without complicating runtime access to the type of that instance (for casting, reflection etc) which is essential in Java language. Since interface cannot define instance fields it does not require a sub-object within an instance of its inherited class and as such a class can inherit from multiple interfaces.