Inheritance

12 April 2022 17:00

Package: It is a named group of related types organized together to avoid collisions between their names and names of other types not belonging to that group and to hide some of them or their members from other types not belonging to that group. A type **T** that belongs to a package **p** has following characteristics

1. It is accessible outside of **p** through its *fully qualified name* of **p.T** only if it is defined with *public* modifier (in T.java).
2. Its binary representation is loaded by default from file **T.class** present in a directory with name **p**.

Visibility of members outside of their declaring type

Access Modifier	**Current Package**	**External Package**
private	none	none
<default>	all	none
protected	all	sub-types
public	all	all

Object Identity: Two objects are considered to be *identical* if they refer to the same instance in the memory. Whether object **x** is identical to object **y** can be determined from expression **x == y**.

Object Equality: Two object are considered to be *equal* if they refer to instances of same class with matching state in the memory. Whether 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 reference type which does not support instantiation but can define instance fields	It is a reference type which cannot define instance fields and as such does not support instantiation
It can define an abstract (unimplemented instance) method using *abstract* modifier	Its methods are implicitly abstract unless defined with *default* modifier

It can contain *public* as well as *non-public* members	It can only contain *public* members
It can define a constructor which can be called by its sub-class	It cannot define a constructor
It can define a static field which may or may not be final	It can only define final static fields
It can extend exactly one other class	It can extend multiple other interfaces
A non-abstract (instantiable) class can inherit from a single abstract class and it must override its abstract methods	A non-abstract class can inherit from multiple interfaces and it must override their abstract methods
Generally defined for segregating common state-associated behavior shared by different type of objects	Generally defined for segregating common state-independent behavior shared by different type of objects

Multiple Inheritance in Java: The type-system of JVM does not allow a class to inherit from multiple classes because an instance of such a class will contain multiple sub-objects (each holding values of fields inherited from a corresponding super-class) which complicates runtime type access required for *safe-casting* and *reflection*. Since an interface cannot define instance-fields it does not require a sub-object within an instance of its inheriting class and as such a class can inherit from multiple interfaces in Java.