Unit –II

Class and Object

Class

<class modifiers> class <class name><formal type parameter list> <extends clause> <implements clause> // Class header

{ // class body

<field declarations>

<method declarations>

<nested class declarations>

<nested interface declarations>

<nested enum declarations>

<constructor declarations>

<initializer blocks>

} // End of Class Body

Simple Class

```
class Cuboid
{
      float length;
      float breadth;
      float height;
}
```

Declaring Objects

- Obtaining objects of a class is a two-step process.
- **First,** you must declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can *refer to an object.*
- ***Second****,* you must acquire an actual, physical copy of the object and assign it to that variable. You can do this using the **new operator. The new operator dynamically allocates (that** is, allocates at run time) memory for an object and returns a reference to it. This reference is, more or less, the address in memory of the object allocated by **new.**

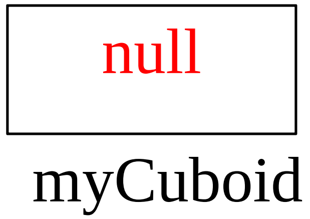
Examle of Declaring Object

- First Step
  - Cuboid myCuboid; // declare reference to object
- Second Step
  - myCuboid= new Cuboid(); // allocate a Cube object
- We can combine above two steps in single step.
- Cuboid myCuboid = new Cuboid();

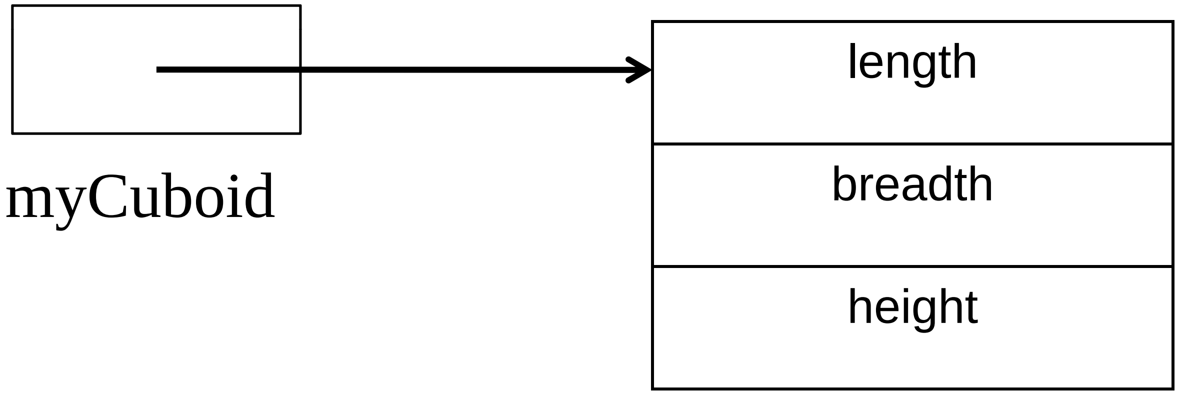
Declaring an object of type Cuboid

Statement **Effect**

Cuboid myCuboid;



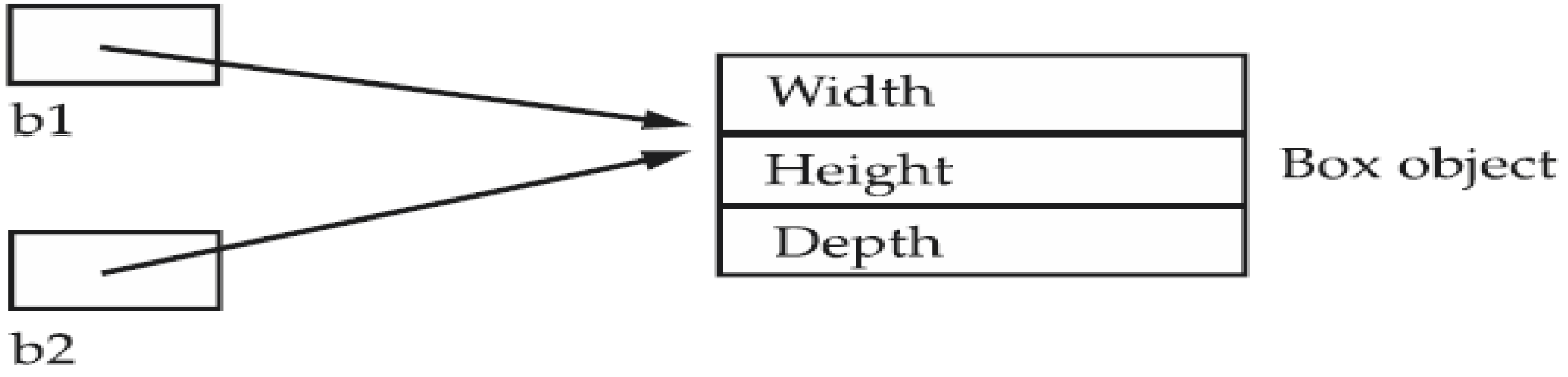
myCuboid = new Cuboid();



Assigning Object Reference Variables

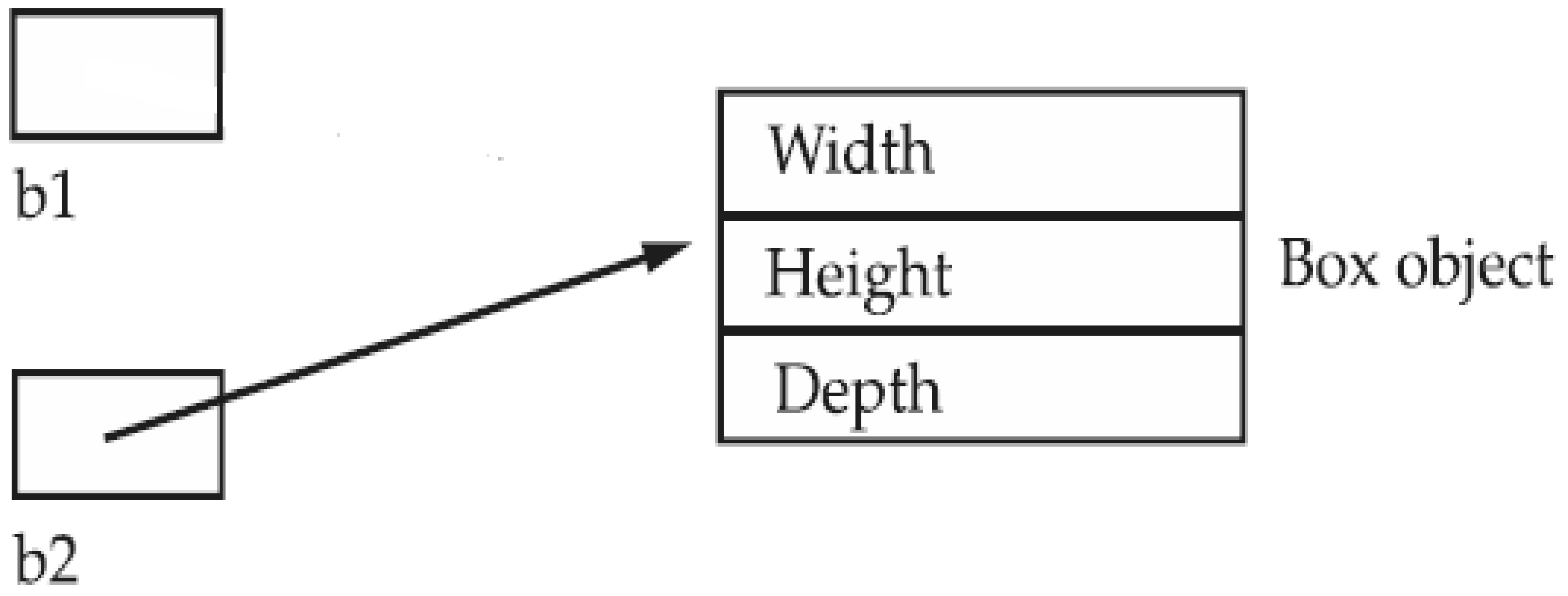
```
class Box {
        double width;
        double height;
        double depth;
        }
```

- Box b1 = new Box();
- Box b2 = b1;



Assigning Object Reference Variables

- b1 = null;
- Here, **b1 has been set to null, but b2 still points to the original object.**



Static Members

- Static members belong to the class in which they are declared and are not part of any instance of the class.
- The declaration of static members is prefixed by the keyword static to distinguish them from instance members.
- Static members are
  - Static variables
  - Static methods
- **Static variables** (also called *class variables) exist in the class they are defined in only.* They are not instantiated when an instance of the class is created. In other words, the values of these variables are not a part of the state of any object.
- **Static methods** are also known as *class methods. A static method in a class can*
- directly access other static members in the class.

Static Methods

- Methods declared as **static have following restrictions:**

  - They can only call other **static methods.**

  - They must only access **static data.**

  - They cannot refer to **this or super in any way.**

Demonstrate static variables, methods, and blocks

```
class StaticTest
{
      static int a = 3;
      static int b;
      static void disp(int x)
     {
          System.out.println("x = " + x);
          System.out.println("a = " + a);
          System.out.println("b = " + b);
     }
     static
    {
          System.out.println("Static block initialized.");
          b = a * 4;
     }
    public static void main(String args[])
   {
         disp(42);
    }
}
```

Terminology for Class Members

- ❖ **Instance Members** :- These are instance variables and instance methods of an object. They can only be accessed or invoked through an object reference.
- ❖ **Instance Variable**:- A field that is allocated when the class is instantiated, i.e., when an object of the class is created. Also called non-static f*ield.*
- ❖ **Instance Method** :- A method that belongs to an instance of the class. Objects of the same class share its implementation.
- ❖ **Static Members**:- These are static variables and static methods of a class. They can be accessed or invoked either by using the class name or through an object reference.
- ❖ **Static Variable** :- A field that is allocated when the class is loaded. It belongs to the class and not to any specific object of the class. Also called static field or class variable.
- ❖ **Static Method**:- A method which belongs to the class and not to any object of the class. Also called class method.

Final members

- A final variable is a constant despite being called a variable. Its value cannot be changed once it has been initialized.
- Instance and static variables can be declared final.
- The keyword final can also be applied to local variables, including method parameters.
- Declaring a variable final has the following implications:
  - A final variable of a primitive data type cannot change its value once it has been initialized.
  - A final variable of a reference type cannot change its reference value once it has been initialized. This effectively means that a final reference will always refer to the same object. However, the keyword final has no bearing on whether the *state of the object denoted by the reference can be changed or not.*

Example of final members

- final int FILE_NEW = 1;
- final int FILE_OPEN = 2;
- final int FILE_SAVE = 3;
- final int FILE_SAVEAS = 4;
- final int FILE_QUIT = 5;

Review Question

Q.1 Which one of these declarations is a valid method declaration?

Select the one correct answer.

(a) void method1 { /* ... */ }

(b) void method2() { /* ... */ }

(c) void method3(void) { /* ... */ }

(d) method4() { /* ... */ }

(e) method5(void) { /* ... */ }

Answer: (b)

Only (b) is a valid method declaration. Methods must specify a return type or must be declared void. This makes (d) and (e) invalid. Methods must specify a list of zero or more comma-separated parameters enclosed by parentheses, ( ). The keyword void cannot be used to specify an empty parameter list. This makes (a) and (c) invalid.

Constructor

- Constuctor is a special method in class
- The main purpose of constructors is to set the initial state of an object, when the object is created by using the new operator.
- Constructors cannot return a value and, therefore, do not specify a return type, not even void, in the constructor header. But their declaration can use the return statement that does not return a value in the constructor body.
- The constructor name must be the same as the class name.
- Contructor would be
  - Default Constructor
  - Overloaded or prameterized Consturctor

General Syntax of Constructor

```
<accessibility modifier> <class name> (<formal parameter list>)<throws clause>
{ // Constructor body
    <local variable declarations>
    <nested local class declarations>
    <statements>
}
```

Default Constructor

- A *default constructor is a constructor without any parameters, i.e., it is a no-parameter* constructor. It has the following signature:
  - *<class name>() { //statements}*
- If a class does not specify *any constructors, then an implicit default constructor is* generated for the class by the compiler. The implicit default constructor is equivalent to the following implementation:
  - *<class name>() { super(); } // No parameters. Calls superclass constructor.*

Overloaded or Prameterized Consturctor

- Like methods, constructors can also be overloaded.
- The constructors in a class all have the same name as the class, their signatures are differentiated by their parameter lists.

Review Question

Q.2 Given the following pairs of method declarations, which statements are true?

```
(i) void fly(int distance) { }
    int fly(int time, int speed) { return time*speed; }
(ii)void fall(int time) { }
    int fall(int distance) { return distance; }
(iii)void glide(int time) { }
    void Glide(int time) { }
```

Answer: (a) and (d)

Select the two correct answer.

(a) The first pair of methods will compile, and overload the method name fly.

(b) The second pair of methods will compile, and overload the method name fall.

(c) The third pair of methods will compile, and overload the method name glide.

(d) The second pair of methods will not compile.

(e) The third pair of methods will not compile.

Review Question

Q.3 Given a class named Book, which one of these constructor declarations is valid for the class Book?

Select the one correct answer.

(a) Book(Book b) {}

(b) Book Book() {}

(c) private final Book() {}

(d) void Book() {}

(e) public static void Book(String[] args) {}

(f) abstract Book() {}

Answer: (a)

A constructor cannot specify any return type, not even void. A constructor cannot be final, static, or abstract.

Review Question

Q.4 Which statements are true?

Select the two correct answers.

(a) A class must define a constructor.

(b) A constructor can be declared private.

(c) A constructor can return a value.

(d) A constructor must initialize all fields when a class is instantiated.

(e) A constructor can access the non-static members of a class.

Answer: *(b) and (e)*

A constructor can be declared private, but this means that this constructor can only be used within the class. Constructors need not initialize all the fields when a class is instanstiated. A field will be assigned a default value if not explicitly initialized. A constructor is non-static and, as such, it can directly access both the static and non-static members of the class.

Review Question

Q.5 What will be the result of compiling the following program?

```
public class MyClass {
    long var;
    public void MyClass(long param) { var = param; } // (1)
    public static void main(String[] args) {
    MyClass a, b;
    a = new MyClass(); // (2)
    b = new MyClass(5); // (3)  } }
```

Answer: (c)

Select the one correct answer.

(a) A compilation error will occur at (1), since constructors cannot specify a return value.

(b) A compilation error will occur at (2), since the class does not have a default constructor.

(c) A compilation error will occur at (3), since the class does not have a constructor that takes one argument of type int.

(d) The program will compile without errors.

Enumerated Types

- An enumerated type defines a finite set of symbolic names and their values.
- These symbolic names are usually called **enum** constants or named constants.
- The canonical form of declaring an **enum** *type is*
  - enum MachineState { BUSY, IDLE, BLOCKED }
- The keyword **enum** is used to declare an **enum** type.
- The basic notation requires the *type name and a comma-separated list of **enum** constants.*

Example of enum

```
enum MachineState { BUSY, IDLE, BLOCKED }
class Machine {
    private MachineState state;
    public void setState(MachineState state) { this.state = state; }
    public MachineState getState() { return this.state; }
}
public class MachineClient {
    public static void main(String[ ] args) {
      Machine machine = new Machine();
      machine.setState(MachineState.IDLE); // (1) Passed as a value.
      // machine.setState(1); // (2) Compile-time error!
      MachineState state = machine.getState(); // (3) Declaring a reference.
      System.out.println("The machine state is: " + state); // (4) Printing the enum name.
      // MachineState newState = new MachineState();// (5) Compile-time error!
    }
}
```

Declaring enum Constructors and Members

- An **enum** type declaration is a special kind of reference type declaration. It can declare constructors and other members as in an ordinary class, RUNGTA

Review Question

Q. 6 Given the following declaration, which expression returns the size of the array, assuming the array has been initialized?

```
int[] array;
```

Select the one correct answer.

(a) array[].length()

(b) array.length()

(c) array[].length

(d) array.length

(e) array[].size()

(f) array.size()

Answer: *(d)*

In Java, arrays are objects. Each array object has a final field named length that stores the size of the array.

Review Question

Q.7 Is it possible to create arrays of length zero?

Select the one correct answer.

(a) Yes, you can create arrays of any type with length zero.

(b) Yes, but only for primitive data types.

(c) Yes, but only for arrays of reference types.

(d) No, you cannot create zero-length arrays, but the main() method may be passed a zero-length array of Strings when no program arguments are specified.

(e) No, it is not possible to create arrays of length zero in Java.

Answer: *(a)*
Java allows arrays of length zero. Such an array is passed as argument to the main() method when a Java program is run without any program arguments.

Review Question

Q.8 Which one of the following array declaration statements is not legal?

Select the one correct answer.

(a) int []a[] = new int [4][4];

(b) int a[][] = new int [4][4];

(c) int a[][] = new int [][4];

(d) int []a[] = new int [4][];

(e) int [][]a = new int [4][4];

Answer: *(c)* The [] notation can be placed both after the type name and after the variable name in an array declaration. Multidimensional arrays are created by constructing arrays that can contain references to other arrays. The expression new int[4][] will create an array of length 4, which can contain references to arrays of int values. The expression new int[4][4] will create the same two-dimensional array, but will in addition create four more one-dimensional arrays, each of length 4 and of the type int[]. References to each of these arrays are stored in the two-dimensional array.
The expression int[][4] will not work, because the arrays for the dimensions must be created from left to right.

Review Question

Q.9 Which of these array declaration statements are not legal?

Select the two correct answers.

(a) int[] i[] = { { 1, 2 }, { 1 }, {}, { 1, 2, 3 } };

(b) int i[] = new int[2] {1, 2};

(c) int i[][] = new int[][] { {1, 2, 3}, {4, 5, 6} };

(d) int i[][] = { { 1, 2 }, new int[2] };

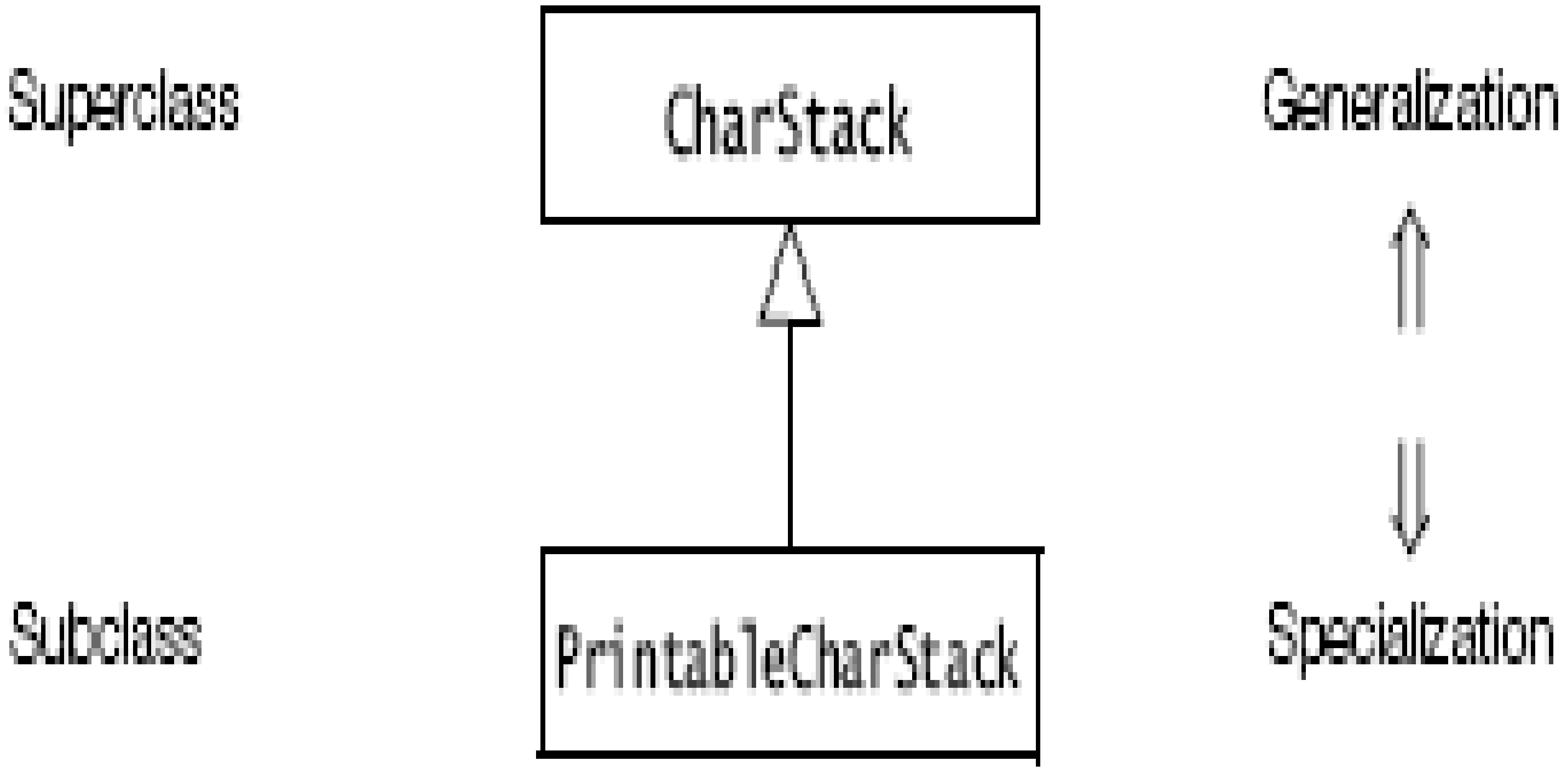
(e) int i[4] = { 1, 2, 3, 4 };

Answer: *(b) and (e)*
The size of the array cannot be specified, as in (b) and (e). The size of the array is given implicitly by the initialization code. The size of the array is never specified in the declaration of an array reference. The size of an array is always associated with the array instance (on the right-hand side), not the array reference (on the lefthand side).

Inheritance

- There are two fundamental mechanisms for building new classes from existing ones: *inheritance and aggregation.*
- *It makes sense to inherit from an existing class* Vehicle to define a class Car, since a car is a vehicle. The class Vehicle has several *parts; therefore, it makes sense to define a composite object of the class Vehicle that* has *constituent objects of such classes as Motor, Axle, and GearBox, which make up a* vehicle.
- *Inheritance is one of the fundamental mechanisms for code reuse in OOP. It allows* new classes to be derived from an existing class. The new class (also called *subclass, subtype, derived class, child class) can inherit members from the old class (also called superclass, supertype, base class, parent class). The subclass can add new behavior and* properties and, under certain circumstances, modify its inherited behavior.

Class Diagram Depicting Inheritance Relationship



Type of Inheritance

- Single Inheritance
- Multilevel Inheritance
- Multiple Inheritance (Java does not support through class)
- Herarical Inheritance
- Hybrid Inheritance

Syntax

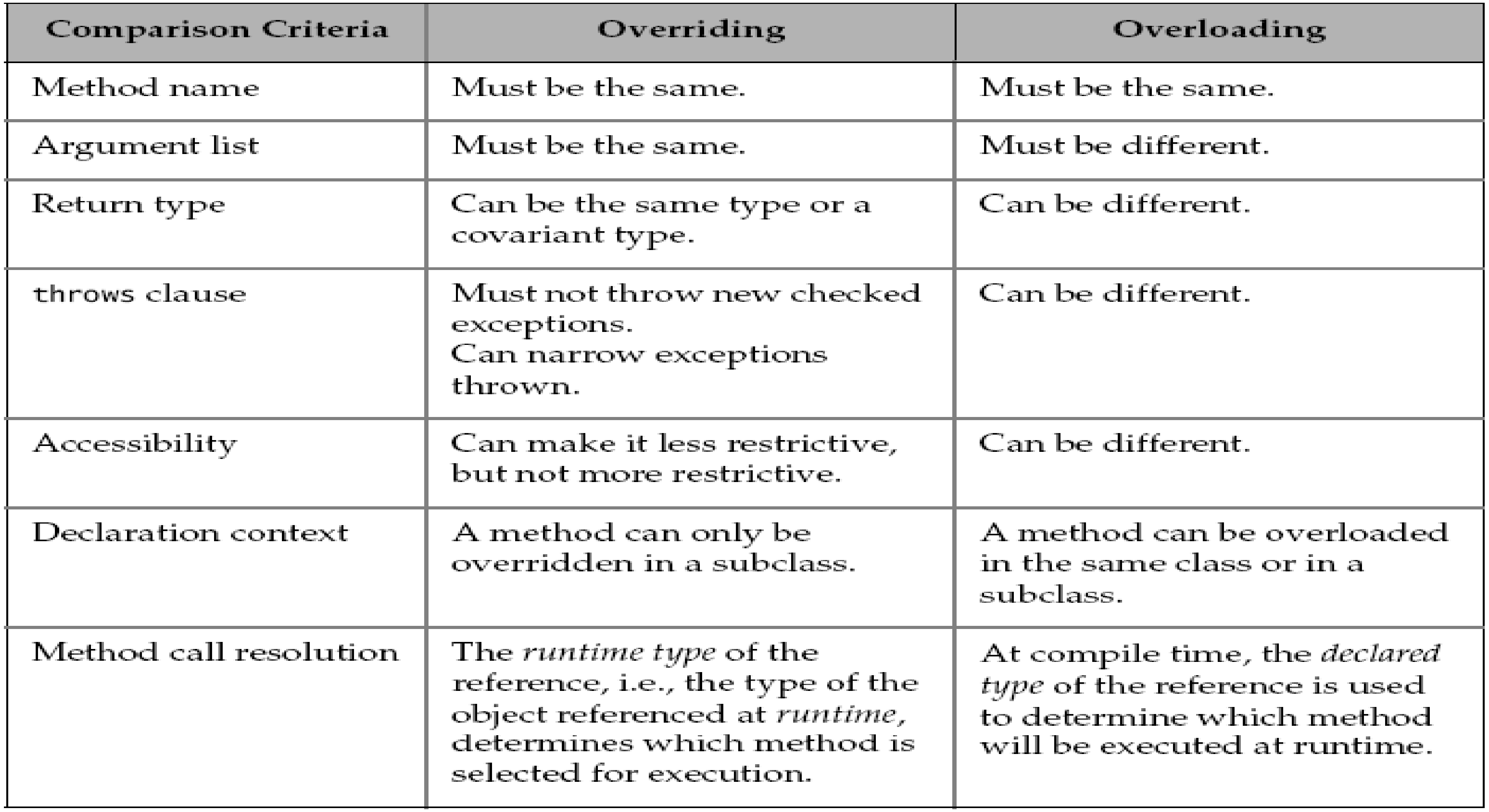
```
class <sub class name> extends <base class name>

{

// class members

}
```

Method Overriding

- When ever super class method signature and subclass method signature both are same it is said to be overriding.

- To perform overriding inheritance is compulsory.

Overriding vs. Overloading



Comparison Criteria	Overriding	Overloading
Method name	Must be the same.	Must be the same.
Argument list	Must be the same.	Must be different.
Return type	Can be the same type or a covariant type.	Can be different.
throws clause	Must not throw new checked exceptions. Can narrow exceptions thrown.	Can be different.
Accessibility	Can make it less restrictive, but not more restrictive.	Can be different.
Declaration context	A method can only be overridden in a subclass.	A method can be overloaded in the same class or in a subclass.
Method call resolution	The *runtime type* of the reference, i.e., the type of the object referenced at *runtime*, determines which method is selected for execution.	At compile time, the *declared type* of the reference is used to determine which method will be executed at runtime.

The Object Reference super

- The keyword super can be used in non-static code (e.g., in the body of an instance method), but only in a subclass, to access fields and invoke methods from the superclass.
- The keyword super provides a reference to the current object as an instance of its superclass.
- super reference is used for member of super class for field member hiding condition.

Example

```
class A {
  int x, y;
void  setValue(int a, int b){
x = a; y = b;
}
 void show() {
    System.out.println (“x“+x+“y“+y); }
class B extends A{
  int x, y;
void  setValue(int a, int b, int c, int d){
super.x = a;
super. y = b;
X = c;
y = d;
}
 void show() {
    System.out.println (“x“+x+“y“+y); }
}
```

The this() Constructor Call

- Constructors cannot be inherited or overridden.
- They can be overloaded, but only in the same class. Since a constructor always has the same name as the class, each parameter list must be different when defining more than one constructor for a class.
- the use of the this() construct, which is used to implement *local chaining of constructors in the class when an instance of the class is created.*
- Java requires that any this() call must occur as the *first statement in a constructor.*

Example of the this() Constructor Call

```
class Light {
// Fields:
        private int noOfWatts;
        private boolean indicator;
        private String location;
// Constructors:
Light() { // (1) Explicit default constructor
        this(0, false);
        System.out.println("Returning from default constructor no. 1.");
}
Light(int watt, boolean ind) { // (2) Non-default
        this(watt, ind, "X");
        System.out.println("Returning from non-default constructor no. 2.");
}
Light(int noOfWatts, boolean indicator, String location) { // (3) Non-default
        this.noOfWatts = noOfWatts;
        this.indicator = indicator;
        this.location = location;
        System.out.println("Returning from non-default constructor no. 3.");
} }
```

The super() Constructor Call

- The super() construct is used in a subclass constructor to invoke a constructor in the *immediate superclass.*
- This allows the subclass to influence the initialization of its inherited state when an object of the subclass is created.
- A super() call in the constructor of a subclass will result in the execution of the relevant constructor from the superclass, based on the signature of the call.
- The super() construct has the same restrictions as the this() construct: if used, the super() call must occur as the *first statement in a constructor, and it can only be* used in a constructor declaration.
- This implies that this() and super() calls cannot both occur in the same constructor.

Review Question

Q. 10 Which constructors can be inserted at (1) in MySub without causing a compile-time error?

```
class MySuper {
        int number;
        MySuper(int i) { number = i; }
}
class MySub extends MySuper {
        int count;
        MySub(int count, int num) {
        super(num);
        this.count = count;
}
// (1) INSERT CONSTRUCTOR HERE
}
```

Select the one correct answer.

(a) MySub() {}
(b) MySub(int count) { this.count = count; }
(c) MySub(int count) { super(); this.count = count; }
(d) MySub(int count) { this.count = count; super(count);
(e) MySub(int count) { this(count, count); }
(f) MySub(int count) { super(count); this(count, 0); }

Answer (e)

The class MySuper does not have a default constructor. This means that constructors in subclasses must explicitly call the superclass constructor and provide the required parameters. The supplied constructor accomplishes this by calling super(num) in its first statement. Additional constructors can accomplish this either by calling the superclass constructor directly using the super() call, or by calling another constructor in the same class using the this() call which, in turn, calls the superclass constructor. (a) and (b) are not valid, since they do not call the superclass constructor explicitly. (d) fails, since the super() call must always be the first statement in the constructor body. (f) fails, since the super() and this() calls cannot be combined.

Review Question

Q. 11 What will the following program print when run?

```
public class MyClass {
public static void main(String[] args) {
B b = new B("Test");
} }
class A {
A() { this("1", "2"); }
A(String s, String t) { this(s + t); }
A(String s) { System.out.println(s); }
}
class B extends A {
B(String s) { System.out.println(s); }
B(String s, String t) { this(t + s + "3"); }
B() { super("4"); }
}
```

Answer: (d)

Select the one correct answer.
(a) It will just print Test.
(b) It will print Test followed by Test.
(c) It will print 123 followed by Test.
(d) It will print 12 followed by Test.
(e) It will print 4 followed by Test.

Review Question

Q. 12 Given the following classes and declarations, which statements are true?

```
// Classes
class Foo {
        private int i;
        public void f() { /* ... */ }
        public void g() { /* ... */ }
}
class Bar extends Foo {
        public int j;
        public void g() { /* ... */ }
}
// Declarations:
        Foo a = new Foo();
        Bar b = new Bar();
```

Select the three correct answers.

(a) The Bar class is a subclass of Foo.
(b) The statement b.f(); is legal.
(c) The statement a.j = 5; is legal.
(d) The statement a.g(); is legal.
(e) The statement b.i = 3; is legal.

Answer (a), (b), and (d)

Bar is a subclass of Foo that overrides the method g(). The statement a.j = 5 is not legal, since the member j in the class Bar cannot be accessed through a Foo reference. The statement b.i = 3 is not legal either, since the private member i cannot be accessed from outside of the class Foo.

Review Question

Q. 13 What would be the result of compiling and running the following program?

```
public class MyClass {
public static void main(String[] args) {
C c = new C();
System.out.println(c.max(13, 29));
}
}
class A {
        int max(int x, int y) { if (x>y) return x; else return y; }
}
class B extends A{
        int max(int x, int y) { return super.max(y, x) - 10; }
}
class C extends B {
        int max(int x, int y) { return super.max(x+10, y+10); } }
```

Select the one correct answer.

(a) The code will fail to compile because the max() method in B passes the arguments in the call super.max(y, x) in the wrong order.

(b) The code will fail to compile because a call to a max() method is ambiguous.

(c) The code will compile and print 13, when run.

(d) The code will compile and print 23, when run.

(e) The code will compile and print 29, when run.

(f) The code will compile and print 39, when run.

Answer (e)

The code will compile without errors. None of the calls to a max() method are ambiguous. When the program is run, the main() method will call the max() method in C with the parameters 13 and 29. This method will call the max() method in B with the parameters 23 and 39. The max() method in B will in turn call the max() method in A with the parameters 39 and 23. The max() method in A will return 39 to the max() method in B. The max() method in B will return 29 to the max() method in C. The max() method in C will return 29 to the main() method.

Package

- A package in Java is an encapsulation mechanism that can be used to group related classes, interfaces, enums, and subpackages.
- Java uses file system directories to store packages.
- At most one package declaration can appear in a source file, and it must be the first statement in the source file.
- The package name is saved in the Java byte code for the types contained in the package.
- If a package declaration is omitted in a compilation unit, the Java byte code for the declarations in the compilation unit will belong to an *unnamed package (also called* the *default package), which is typically synonymous with the current working directory* on the host system.

Defining Packages

- The package statement has the following syntax:
  - **package <*fully qualified package name*>;**
- For example, a multileveled package declared as
  - ***Package mypackage;***
  - needs to be stored in **mypackage**

- The general form of a multileveled package statement is shown here:
  - **package *pkg1[.pkg2[.pkg3]];***
- For example, a multileveled package declared as
  - **package java.awt.image;**
  - needs to be stored in **java\awt\image**

Example

```
package mypack;
class Balance {
        String name;
        double bal;
        Balance(String n, double b) {
        name = n;
        bal = b;
}
void show() {
        if(bal<0)
        System.out.print("--> ");
        System.out.println(name + ": $" + bal);
}
}
class AccountBalance {
public static void main(String args[]) {
        Balance current[] = new Balance[3];
        current[0] = new Balance("K. J. Fielding", 123.23);
        current[1] = new Balance("Will Tell", 157.02);
        current[2] = new Balance("Tom Jackson", -12.33);
        for(int i=0; i<3; i++) current[i].show();
}
}
```

Compile and Run the program

- Create a folder **mypack** andsave java source code AccountBalance.java
- Compile the java source inside mypack
  - **javac AccountBalance.java**
- Make sure that the resulting **.class file is also in the MyPack** directory
- you will need to be in the directory above **mypack when you execute** following command
  - **Java mypack.AccountBalance**
- **AccountBalance is now part of the package mypack. This means that** it cannot be executed by itself. That is, you cannot use this command line:
  - **java AccountBalance**

Using Packages

- We can use liberay package as well user define pakcage
- The simple form of the import declaration has the following syntax:
  - **import <*fully qualified type name*>*;***
  - **import *pkg1[.pkg2].(classname*|**);***
- Example
  - **Importing all classes**
  - **Import java.awt.*;**
  - **Import java.util.*;**
  - **Importing specific class**
  - **import java.util.Date;**
  - **Import java.util.Scanner;**

Using Packages Example

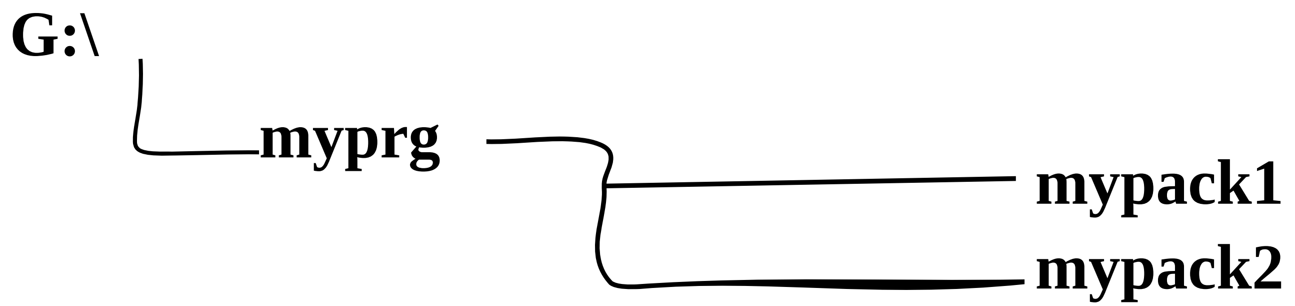
```
import java.util.Scanner;
class  TestInput{
public static void main(String args[]){
String name;
int age;
float fee;
Scanner input = new Scanner(System.in);
System.out.println(“Enter your name :“);
name = input.nextLine();
System.out.println(“Enter your age:“);
age = input.nextInt();
System.out.println(name+“    “ +age);
}}
```

Example to access class of another package

```
//Save A.java inside folder mypack1
package mypack1;
public class A {
 public void show(){
 System.out.println("I am class A");
} }
//Save B.java inside folder mypack2
package mypack2;
import  mypack1.*;
public class B extends A{
  public void show(){
 super.show();
 System.out.println("I am class B");
}
public static void main(String args[]){
    new B().show();
}}
```

Compile and Run the program

Let us my computer has following direcory(package) structure



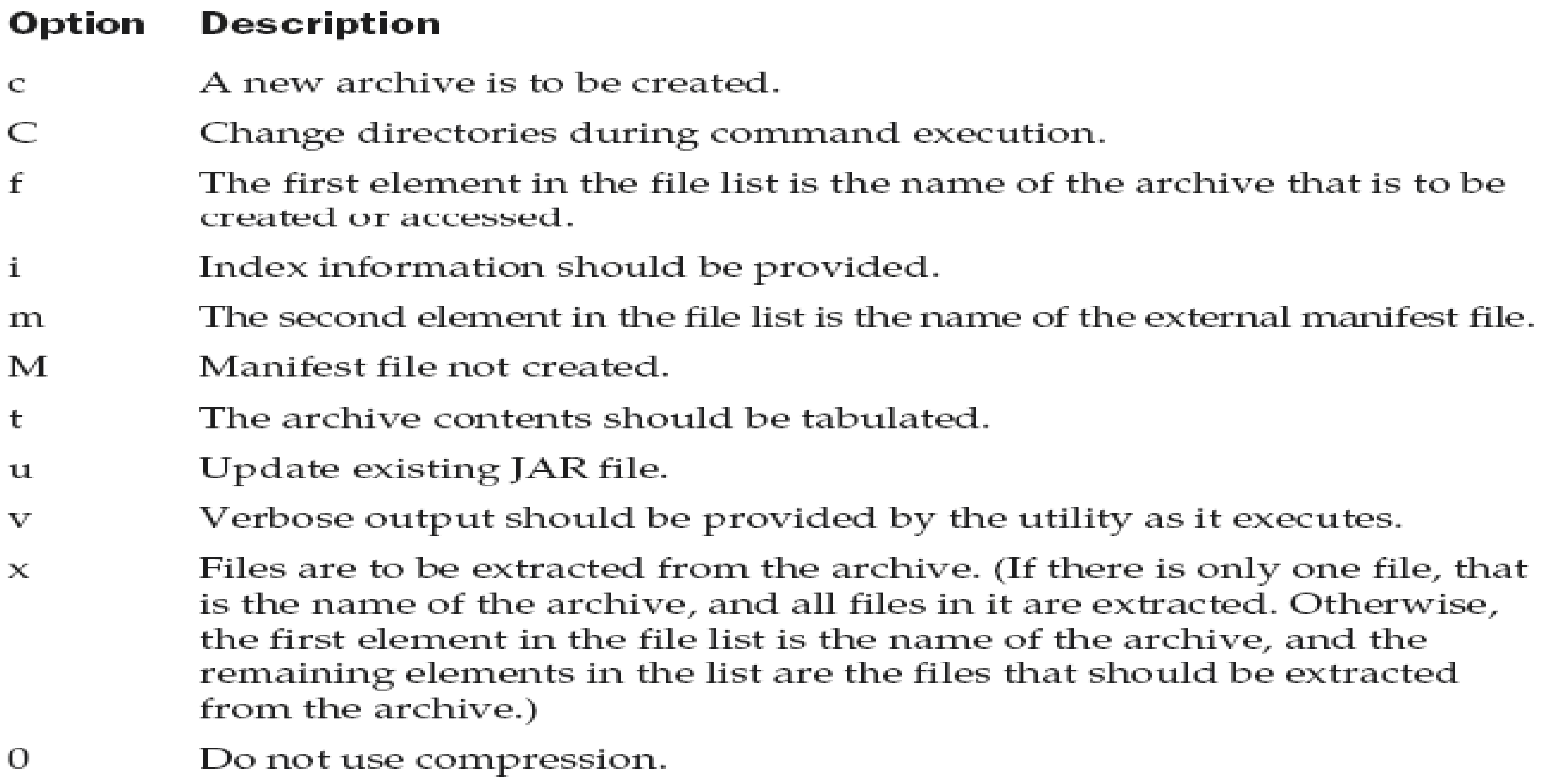
- **Compile**
  - G:\myprg\mypack1> javac A.java
  - G:\myprg\mypack2> javac –classpath /myprg B.java
- **Run**
  - G:\myprg> java mypack2.B

JAR Files

- The JAR (**Java ARchive**) utility provides a convenient way of bundling and deploying Java programs.
- A JAR file is created by using the jar tool.
- A typical JAR file for an application will contain the class files and any other resources needed by the application (for example image and audio files).
- JAR technology makes it much easier to deliver and install software.
- A utility is used to generate a JAR file. Its syntax is shown here:

  - ***jar options files***

JAR Command Options



Option	Description
c	A new archive is to be created.
C	Change directories during command execution.
f	The first element in the file list is the name of the archive that is to be created or accessed.
i	Index information should be provided.
m	The second element in the file list is the name of the external manifest file.
M	Manifest file not created.
t	The archive contents should be tabulated.
u	Update existing JAR file.
v	Verbose output should be provided by the utility as it executes.
x	Files are to be extracted from the archive. (If there is only one file, that is the name of the archive, and all files in it are extracted. Otherwise, the first element in the file list is the name of the archive, and the remaining elements in the list are the files that should be extracted from the archive.)
0	Do not use compression.

Create a JAR file

- **jar -cf Xyz.jar *.class *.gif**
  - Above uses of jar utility create jar file and include all class file and gif inside jar file

- **jar -cf Xyz.jar A.class**
  - Above uses of jar utility create jar file and include only A.class file inside jar file

Review Question

Q.14 What will be the output.

```
public class TestLoop {
public static void main(String[] args) {
int x[] = {1,2,3,4,5};
for (int i:x){
System.out.print(i);
} } }
```

Select correct answer

a) Compile time error

b) Run time error

c) It will compile and run. It will print 1 2 3 4 5

d) It will run and throw exception

Answer: (c)

Access Protection

- Classes and packages are both means of encapsulating and containing the name space and scope of variables and methods.
- The class is Java's smallest unit of abstraction. Because of the interplay between classes and packages.
- Java addresses four categories of visibility for class members:
  - Subclasses in the same package
  - Non-subclasses in the same package
  - Subclasses in different packages
  - Classes that are neither in the same package nor subclasses
- The accessibility of members can be one of the following:
  - **public**
  - **protected**
  - **default (also called *package accessibility)***
  - **private**

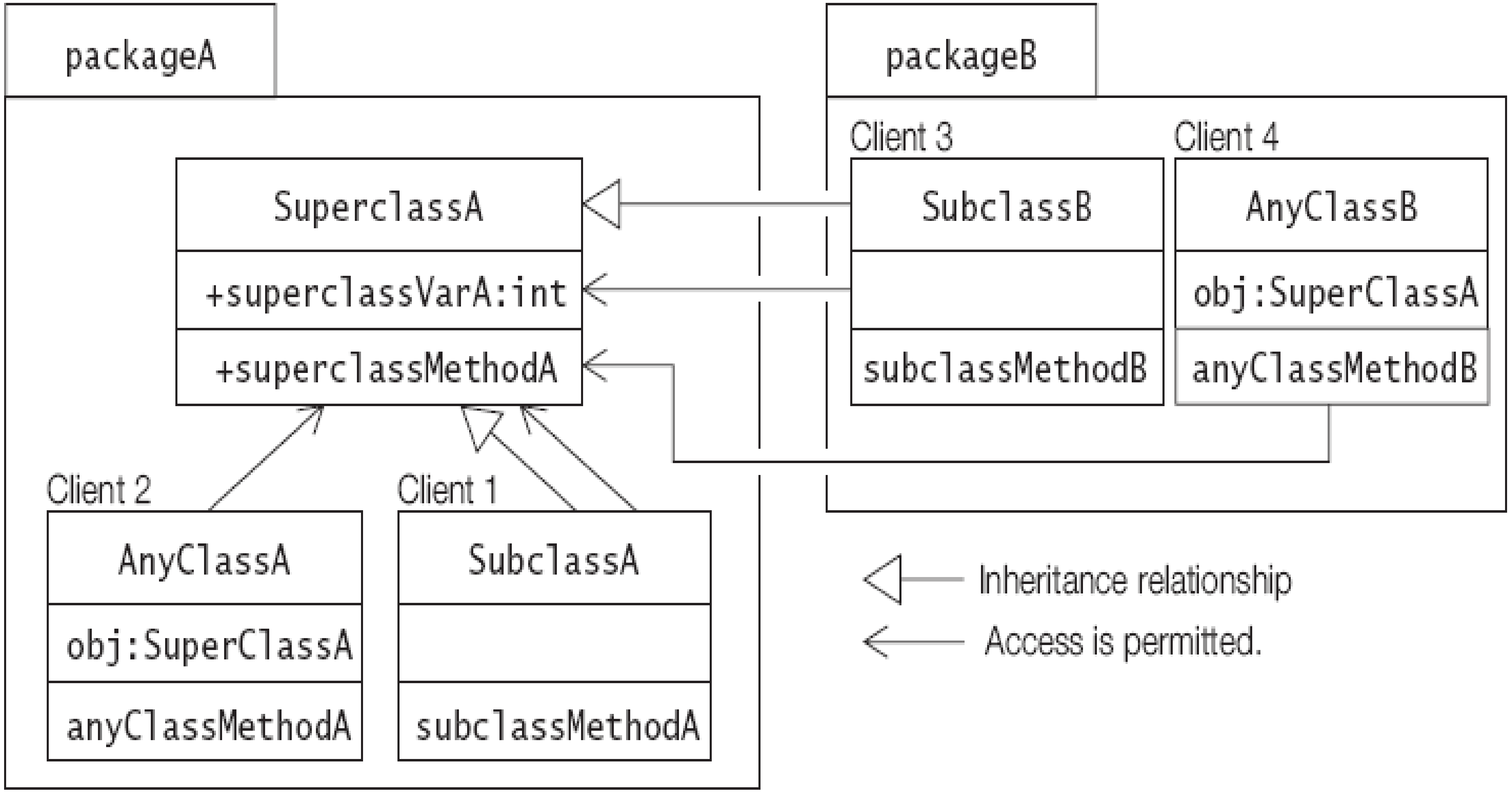
Public member

- Public accessibility is the least restrictive of all the accessibility modifiers.
- A public member is accessible from anywhere, both in the package containing its class and in other packages where this class is visible.
- This is true for both instance and static members.

Example

```
//Filename: SuperclassA.java (1)
package packageA;
public class SuperclassA {
        public int superclassVarA; // (2)
        public void superclassMethodA() {/*...*/} // (3)
}
class SubclassA extends SuperclassA {
        void subclassMethodA() { superclassVarA = 10; } // (4) OK.
}
class AnyClassA {
        SuperclassA obj = new SuperclassA();
        void anyClassMethodA() {
        obj.superclassMethodA(); // (5) OK.
} }
//Filename: SubclassB.java (6)
package packageB;
import packageA.*;
public class SubclassB extends SuperclassA {
        void subclassMethodB() { superclassMethodA(); } // (7) OK.
}
class AnyClassB {
        SuperclassA obj = new SuperclassA();
        void anyClassMethodB() {
        obj.superclassVarA = 20; // (8) OK.
}}
```

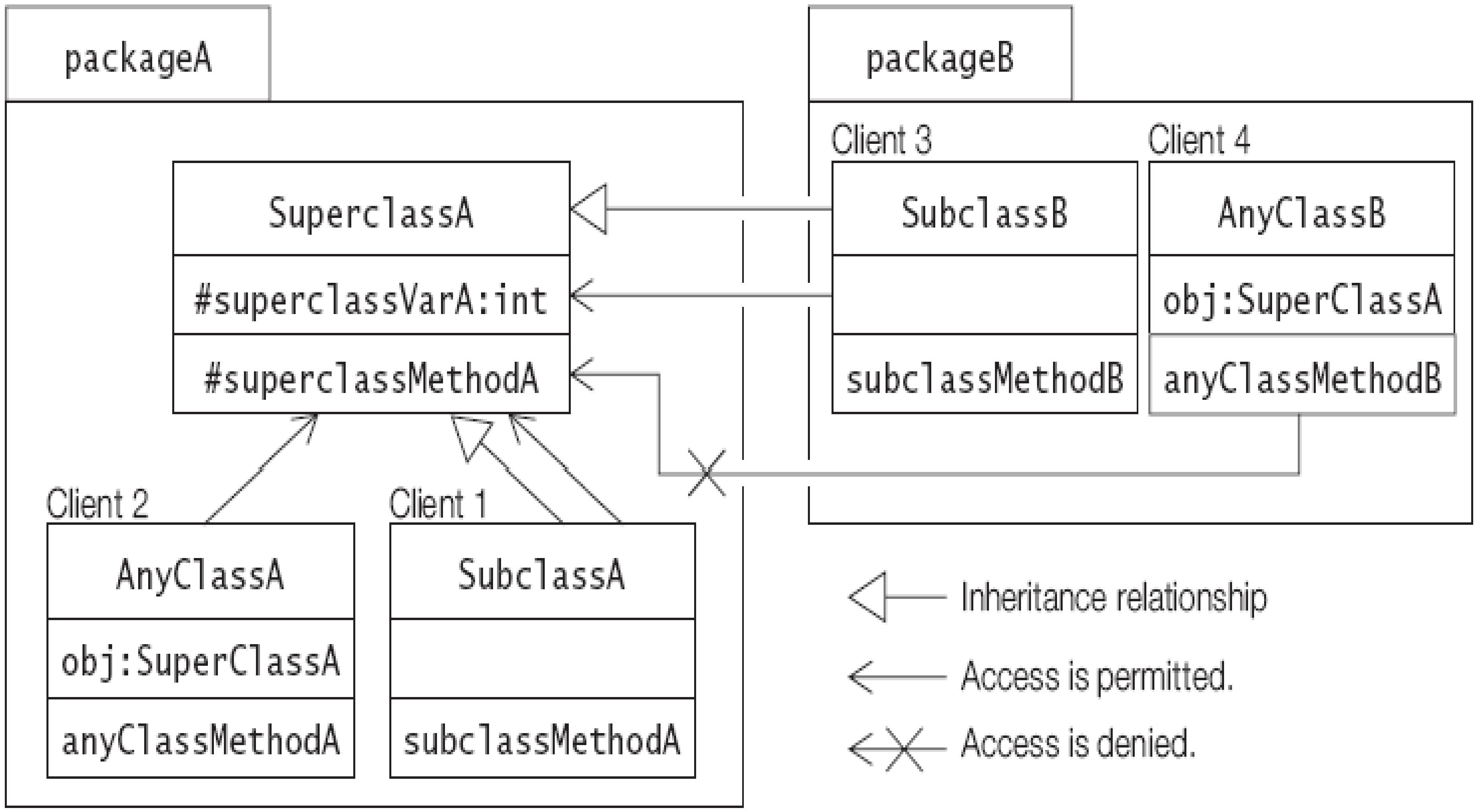
Public Accessibility



Protected member

- A protected member is accessible in all classes in the same package, and by all subclasses of its class in any package where this class is visible.
- In other words, non subclasses in other packages cannot access protected members from other packages.
- It is more restrictive than public member accessibility.

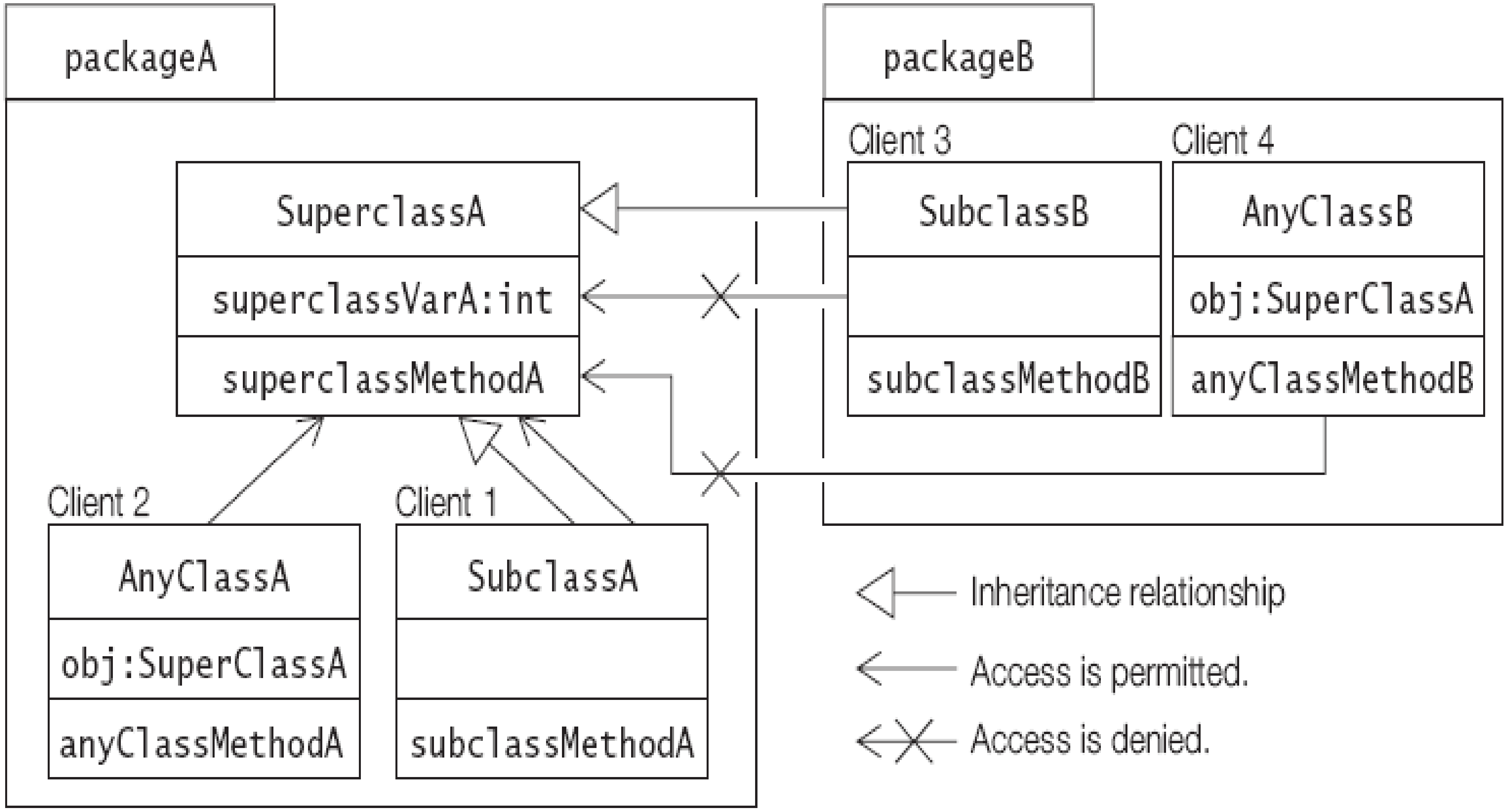
Protected Accessibility



Default Accessibility for Members

- When no member accessibility modifier is specified, the member is only accessible by other classes in its own class's package.
- Even if its class is visible in another (possibly nested) package, the member is not accessible elsewhere.
- Default member accessibility is more restrictive than protected member accessibility.

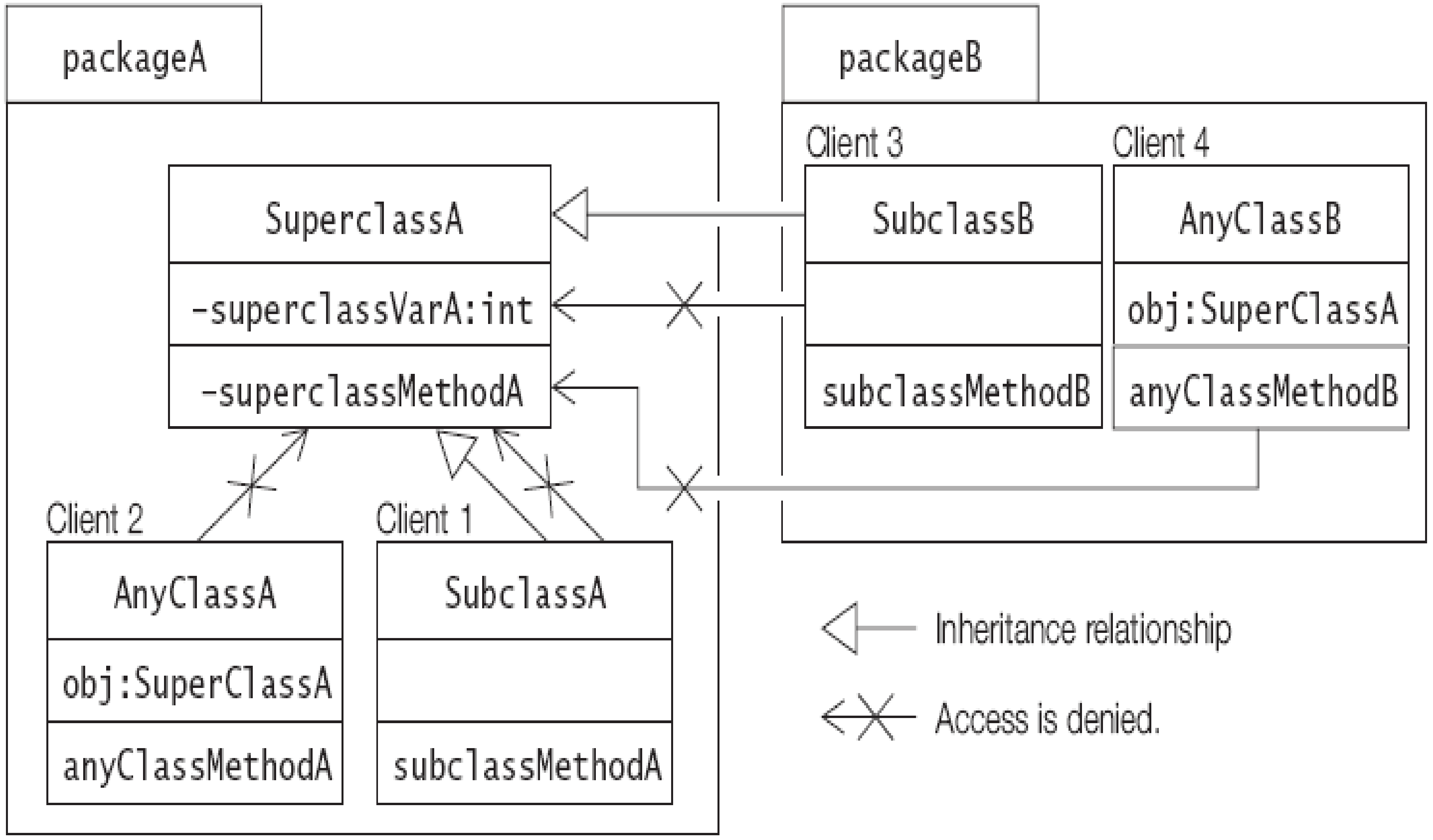
Default Accessibility



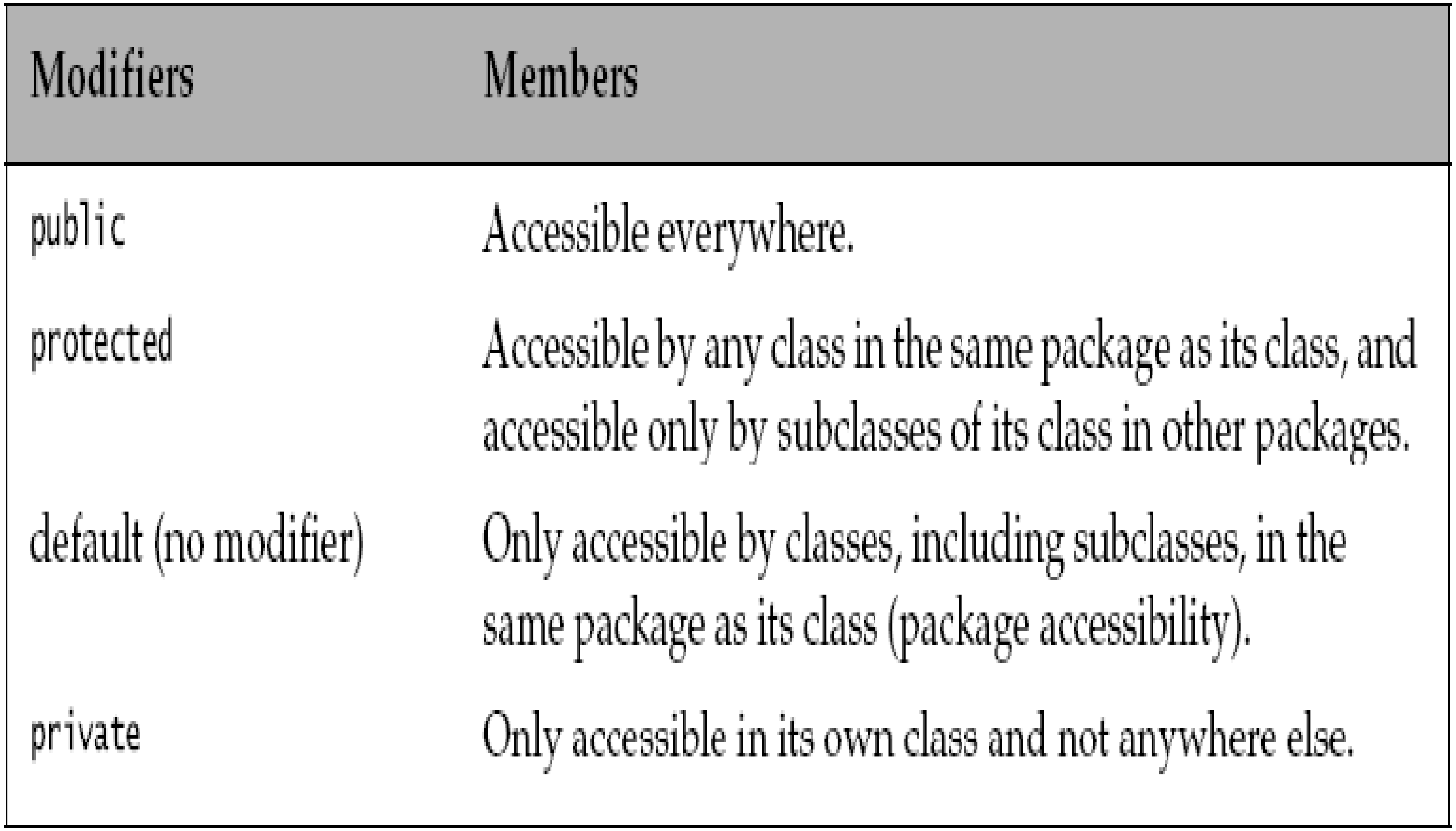
private Members

- This is the most restrictive of all the accessibility modifiers.
- Private members are not accessible from any other classes. This also applies to subclasses, whether they are in the same package or not.
- Since they are not accessible by their simple name in a subclass, they are also not inherited by the subclass.

Private Accessibility

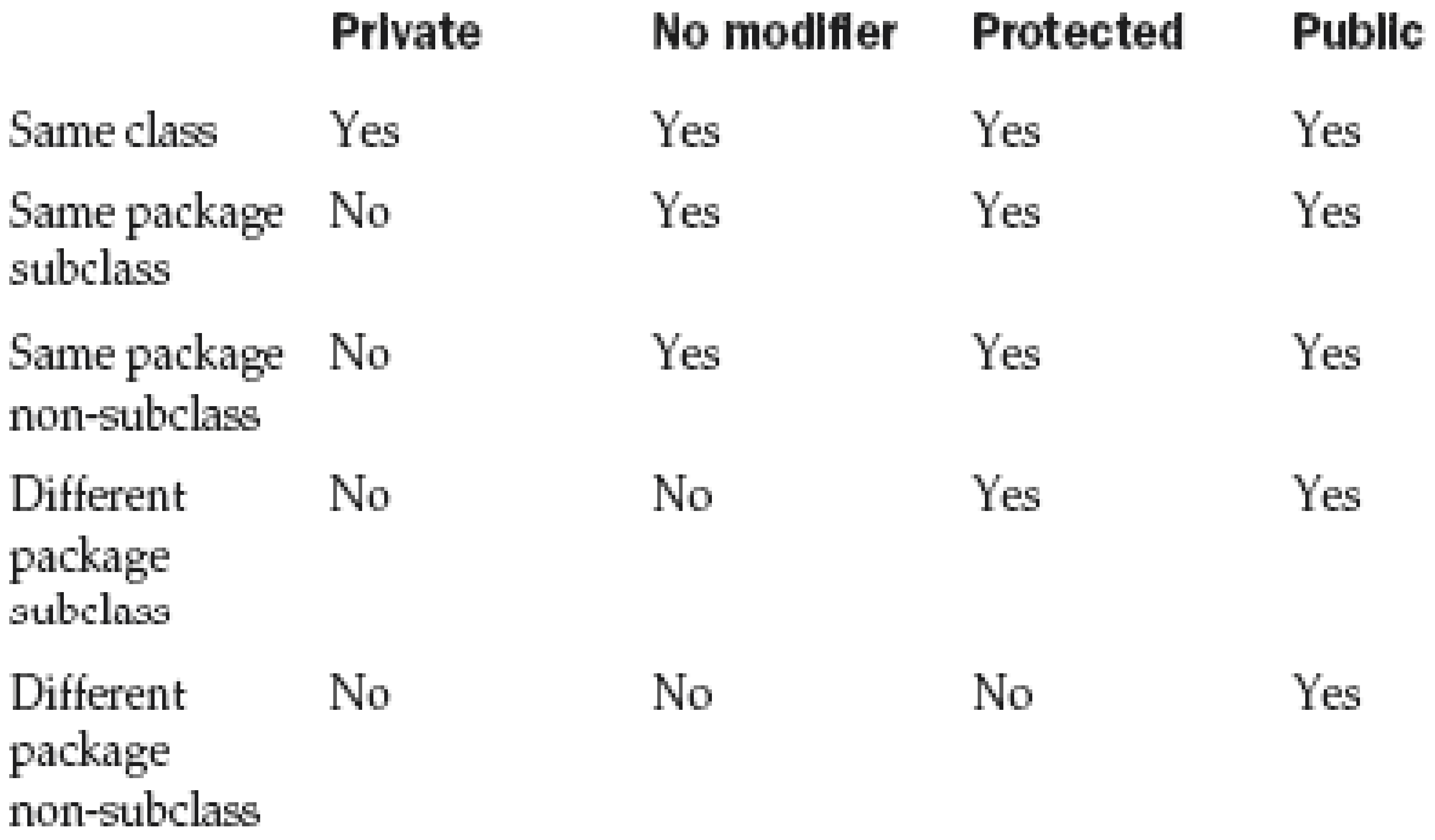


Summary of Accessibility Modifiers for Members



Modifiers	Members
`public`	Accessible everywhere.
`protected`	Accessible by any class in the same package as its class, and accessible only by subclasses of its class in other packages.
default (no modifier)	Only accessible by classes, including subclasses, in the same package as its class (package accessibility).
`private`	Only accessible in its own class and not anywhere else.

Summary of Accessibility Modifiers for Members…



	Private	No modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Review Question

Q.14 Given the following declaration of a class, which fields are accessible from outside the package com.corporation.project?

```
package com.corporation.project;
public class MyClass {
    int i;
    public int j;
    protected int k;
    private int l;
}
```

Select the two correct answers.

(a) Field i is accessible in all classes in other packages.

(b) Field j is accessible in all classes in other packages.

(c) Field k is accessible in all classes in other packages.

(d) Field k is accessible in subclasses only in other packages.

(e) Field l is accessible in all classes in other packages.

(f) Field l is accessible in subclasses only in other packages.

Answer: *(b) and (d)*

Outside the package, the member j is accessible to any class, whereas the member k is only accessible to subclasses of MyClass.

Review Question

Q.15 How restrictive is the default accessibility compared to public, protected, and private accessibility?

Select the one correct answer.

(a) Less restrictive than public.

(b) More restrictive than public, but less restrictive than protected.

(c) More restrictive than protected, but less restrictive than private.

(d) More restrictive than private.

(e) Less restrictive than protected from within a package, and more restrictive than protected from outside a package.

Answer: *(c)*

Review Question

Q. 16 Which statement is true about the accessibility of members?

Select the one correct answer.

(a) A private member is always accessible within the same package.

(b) A private member can only be accessed within the class of the member.

(c) A member with default accessibility can be accessed by any subclass of the class in which it is declared.

(d) A private member cannot be accessed at all.

(e) Package/default accessibility for a member can be declared using the keyword default.

Answer:*(b)*

A private member is only accessible within the class of the member.

Review Question

Q.17 Which lines that are marked will compile in the following code?

```
//Filename: A.java
package packageA;
public class A {
protected int pf;
}
//Filename: B.java
package packageB;
import packageA.A;
public class B extends A {
void action(A obj1, B obj2, C obj3) {
pf = 10; // (1)
obj1.pf = 10; // (2)
obj2.pf = 10; // (3)
obj3.pf = 10; // (4)
} } }
```

```
class C extends B {
void action(A obj1, B obj2, C obj3) {
pf = 10; // (5)
obj1.pf = 10; // (6)
obj2.pf = 10; // (7)
obj3.pf = 10; // (8)
}
}
class D {
void action(A obj1, B obj2, C obj3) {
pf = 10; // (9)
obj1.pf = 10; // (10)
obj2.pf = 10; // (11)
obj3.pf = 10; // (12)
}
```

Select the five correct answers.

(a) (1) (b) (2) (c) (3) (d) (4) (e) (5) (f) (6) (g) (7) (h) (8) (i)(9) (j) (10) (k) (11) (l) (12)

Answer: *(a), (c), (d), (e), and (h)*

Abstract Class

- A class can be declared with the keyword **abstrac**t to indicate that it cannot be instantiated.
- Define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method.
- An **abstract class** is considered incomplete
- Enum types *cannot be declared abstract,*
- **Syntax**

```
abstract class <class name>{

}
```

Abstract Method

- An abstract method does not have an implementation; i.e., no method body is defined for an abstract method, only the *method header is provided in the class declaration.*
- The keyword abstract is mandatory in the header of an abstract method declared in a class.
- Its class is then incomplete and must be explicitly declared abstract.
- Subclasses of an abstract class must then provide the method implementation; otherwise, they must also be declared abstract.
- The accessibility of an abstract method declared in a class cannot be private, as subclasses would not be able to override the method and provide an implementation.
- Only an instance method can be declared abstract. Since static methods cannot be overridden, declaring an abstract static method makes no sense.

An abstract method syntax

- abstract *<accessibility modifier> <return type> <method name> (<parameter list>) <throws clause>;*

Example

```
// A Simple demonstration of abstract.
abstract class A {
abstract void callme();
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
} }
class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
} }
class AbstractDemo {
public static void main(String args[]) {
B b = new B();
b.callme();
b.callmetoo();
} }
```

Final Class

- A class can be declared final to indicate that it cannot be extended; that is, one cannot declare subclasses of a final class.
- In other words, the class behavior cannot be changed by extending the class.
- A final class must be complete, whereas an abstract class is considered incomplete.
- Classes cannot be both final and abstract at the same time.
- **Syntax**

```
final class <class name>{

}
```

Final Method

- A final method in a class is *complete (that is, has an implementation) and cannot be* overridden in any subclass

Interface

- Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without any body.
- Using the keyword **interface, you can fully abstract a class' interface from its implementation.**
- That is, using **interface, you can specify what a class must do, but not how** it does it.
- Once interface is defined, any number of classes can implement it. Also, one class can implement any number of interfaces.
- Interfaces are designed to support dynamic method resolution at run time.

Defining Interfaces

- A top-level interface has the following general syntax:

// *Interface header*

*<**accessibility modifier**> **interface** <**interface name**><**extends interface clause**>*

```
    { // Interface body
    <constant declarations>
    <abstract method declarations>
    <nested class declarations>
    <nested interface declarations>
    }
```

Example of interface

```
interface IStack {

    int top = 10;

    void push(Object item);

    Object pop();

}
```

Methods in interface

- An interface defines a *contract by specifying a set of abstract method declarations,* but provides no implementations.
- The methods in an interface are all implicitly abstract and public by virtue of their definition.
- An abstract method declaration has the following form:
  - ***return-type method-name1(parameter-list)*** *<**throws clause**>;*

Variable in interface

- Variables can be declared inside of interface declarations.
- Variables are implicitly **final** and **static,** meaning they cannot be changed by the implementing class.
- Variables must also be initialized with a constant value.
- Variables are implicitly **public.** It does not accept other accessibility modifier except ***public.***

Implementing Interfaces

- Once an **interface has been defined, one or more classes can implement that interface.**

- The general form of a class that includes the **implements clause :**

access *class* ***classname [****extends* ***superclass]*** **[**implements ***interface [,interface...]] {***

// class-body

}

Example for Variables in Interfaces

```
interface Constants {
double PI_APPROXIMATION = 3.14;
String AREA_UNITS = "sq.cm.";
String LENGTH_UNITS = "cm.";
}
public class Client implements Constants {
public static void main(String[] args) {
double radius = 1.5;
// (1) Using direct access:
System.out.printf("Area of circle is %.2f %s%n", PI_APPROXIMATION * radius*radius,
    AREA_UNITS);
// (2) Using fully qualified name:
System.out.printf("Circumference of circle is %.2f %s%n", 2.0 *
    Constants.PI_APPROXIMATION * radius, Constants.LENGTH_UNITS);
} }
```

Extending the Interface

- An interface can extend other interfaces, using the extends clause.
- A subinterface inherits all methods from its superinterfaces, as their method declarations are all implicitly public
- Multiple inheritance hierarchy between interfaces
- **Example**

```
inteface A{
}
interface C{
}
interface B extends A,C{
}
```

Review Question

Q.18 Which statements about interfaces are true?

Select the two correct answers.

(a) Interfaces allow multiple implementation inheritance.

(b) Interfaces can be extended by any number of interfaces.

(c) Interfaces can extend any number of interfaces.

(d) Members of an interface are never static.

(e) Members of an interface can always be declared static.

Answer : (b) and (c)

Interface declarations do not provide any method implementations and only permit multiple interface inheritance. An interface can extend any number of interfaces and can be extended by any number of interfaces. Fields in interfaces are always static, and can be declared static explicitly. Abstract method declarations in interfaces are always non-static, and cannot be declared static.

Review Question

Q.19 Which of these field declarations are legal within the body of an interface?

Select the three correct answers.

(a) public static int answer = 42;

(b) int answer;

(c) final static int answer = 42;

(d) public int answer = 42;

(e) private final static int answer = 42;

Answer : (a), (c) and (d)

Fields in interfaces declare named constants, and are always public, static, and final. None of these modifiers are mandatory in a constant declaration. All named constants must be explicitly initialized in the declaration.

Review Question

Q.20 Which statements about the keywords extends and implements are true?

Select the two correct answers.

(a) The keyword extends is used to specify that an interface inherits from another interface.

(b) The keyword extends is used to specify that a class implements an interface.

(c) The keyword implements is used to specify that an interface inherits from another interface.

(d) The keyword implements is used to specify that a class inherits from an interface.

(e) The keyword implements is used to specify that a class inherits from another class.

Answer : (a) and (d)

Review Question

Q.21 Which statement is true about the following code?

```
interface Interface1 {
int VAL_A = 1;
int VAL_B = 2;
void f();
void g();
}
```

```
interface Interface2 {
int VAL_B = 3;
int VAL_C = 4;
void g();
void h();
}
```

Answer : (d)

```
abstract class MyClass implements Interface1, Interface2 {
        public void f() { }
        public void g() { }
        }
```

Select the one correct answer.

(a) MyClass only implements Interface1. Implementation for void h() from Interface2 is missing.

(b) The declarations of void g() in the two interfaces conflict, therefore, the code will not compile.

(c) The declarations of int VAL_B in the two interfaces conflict, therefore, the code will not compile.

(d) Nothing is wrong with the code, it will compile without errors.

Review Question

Q.22 Which declaration can be inserted at (1) without causing a compilation error?

```
interface MyConstants {
int r = 42;
int s = 69;
// (1) INSERT CODE HERE
}
```

Answer : (a) and (c)

Select the two correct answers.

(a) final double circumference = 2 * Math.PI * r;

(b) int total = total + r + s;

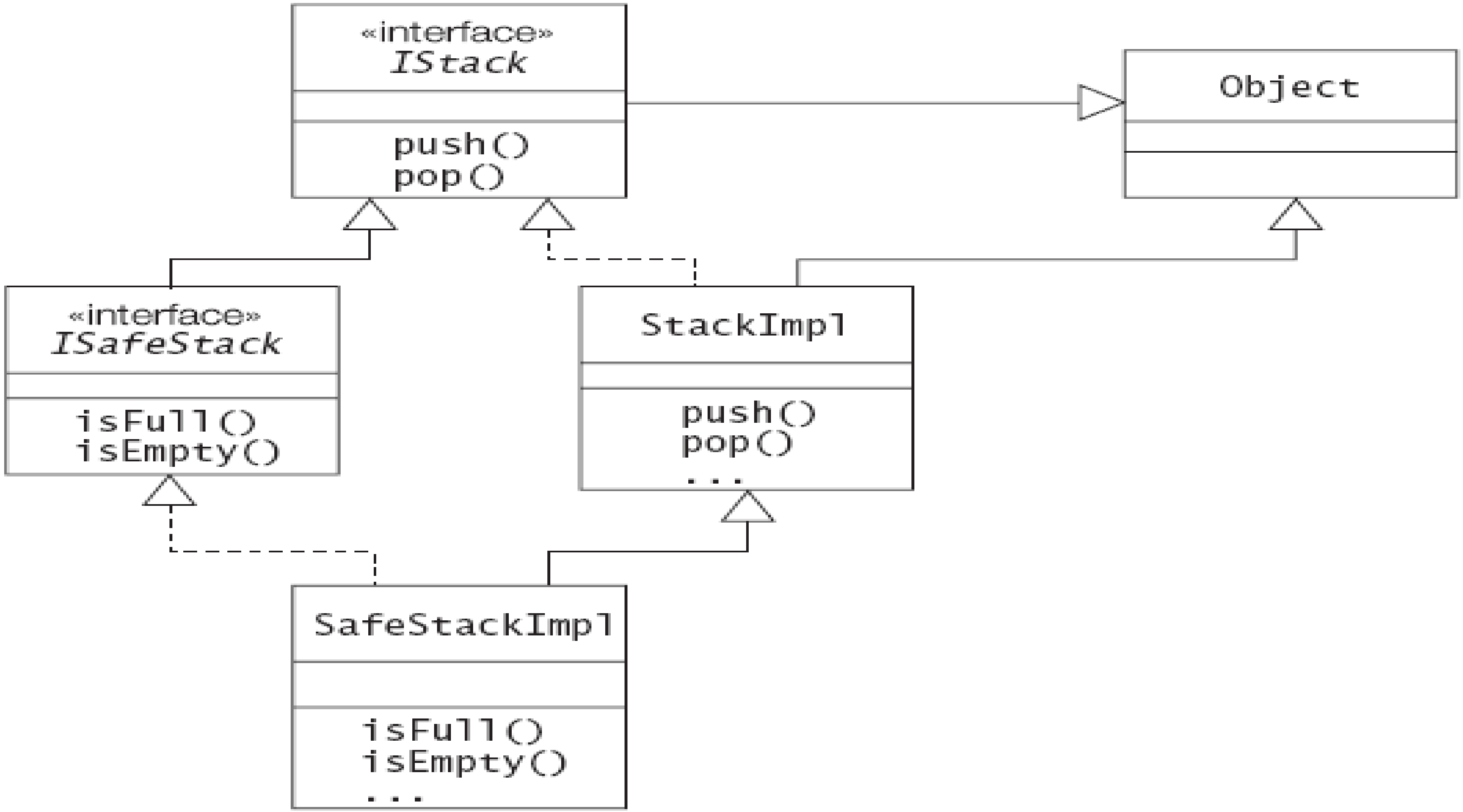
(c) int AREA = r * s;

(d) public static MAIN = 15;

(e) protected int CODE = 31337;

Programming Exercises

- Q. 1 Create a program for following *Inheritance Relations*



Programming Exercises

Q.2 Declare an interface called **Function** that has a method named evaluate that takes an int parameter and returns an int value.

Create a class called **Half** that implements the Function interface. The implementation of the method evaluate() should return the value obtained by dividing the int argument by 2.

In a **client**, create a method that takes an arbitrary array of int values as a parameter, and returns an array that has the same length, but the value of an element in the new array is half that of the value in the corresponding element in the array passed as the parameter. Let the implementation of this method create an instance of Half, and use this instance to calculate values for the array that is returned.

Programming Exercises

Q.3 Rewrite the method that operated on arrays from the previous exercise 2: the method should now also accept a Function reference as an argument, and use this argument instead of an instance of the Half class.

Create a class called Print that implements the method evaluate() in the Function interface. This method simply prints the int value passed as argument, and returns this value.

Now, write a program that creates an array of int values from 1 to 10, and does the following:

- Prints the array using an instance of the Print class and the method described earlier.
- Halves the values in the array and prints the values again, using the Half and Print classes, and the method described above.

Programming Exercises

Q. 4Design a class for a bank database. The database should support the following

operations:

- deposit a certain amount into an account
- withdraw a certain amount from an account
- get the balance (i.e., the current amount) in an account

transfer an amount from one account to another The amount in the transactions is a value of type double. The accounts are identified by instances of the class Account that is in the package myprg.records. The database class should be placed in a package called myprg.mysys The deposit, withdraw, and balance operations should not have any implementation, but allow subclasses to provide the implementation. The transfer operation should use the deposit and withdraw operations to implement the transfer. It should not be possible to alter this operation in any subclass, and only classes within the package myprg.mysys should be allowed to use this operation. The deposit and withdraw operations should be accessible in all packages. The balance operation should only be accessible in subclasses and classes within the package myprg.mysys

Object Finalization

- Object finalization provides an object with a last resort to undertake any action before its storage is reclaimed.
- The automatic garbage collector calls the finalize() method in an object that is eligible for garbage collection before actually destroying the object.
- The **finalize( ) method has this general form:**

```
protected void finalize() [throws Throwable]
{
}
```

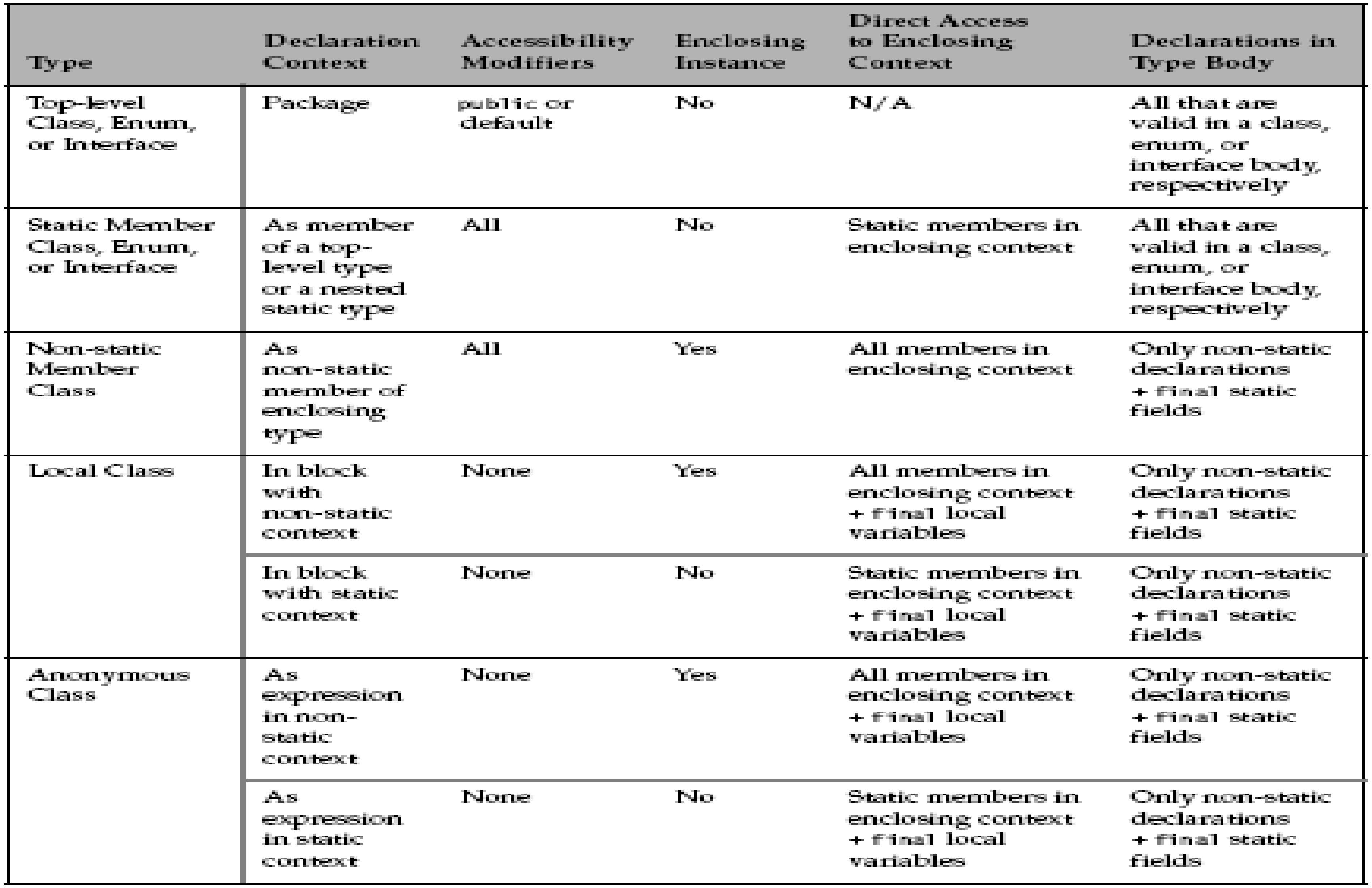
- the keyword **protected is a specifier that prevents access to finalize( ) by code** defined outside its class.

RUNGTA

Nested Class

- A class that is declared within another type declaration is called a *nested class.*
- an interface or an enum type that is declared within another type declaration is called a *nested interface or a nested enum type, respectively*
- there are four categories of *nested classes,*
  - static member classes, enums, and interfaces
  - non-static member classes
  - local classes
  - anonymous classes
- Non static member class, local class and anonymous class are collectively known as ***inner classes.***

Overview of Type Declarations



Type	Declaration Context	Accessibility Modifiers	Enclosing Instance	Direct Access to Enclosing Context	Declarations in Type Body
Top-level Class, Enum, or Interface	Package	public or default	No	N/A	All that are valid in a class, enum, or interface body, respectively
Static Member Class, Enum, or Interface	As member of a top-level type or a nested static type	All	No	Static members in enclosing context	All that are valid in a class, enum, or interface body, respectively
Non-static Member Class	As non-static member of enclosing type	All	Yes	All members in enclosing context	Only non-static declarations + final static fields
Local Class	In block with non-static context	None	Yes	All members in enclosing context + final local variables	Only non-static declarations + final static fields
	In block with static context	None	No	Static members in enclosing context + final local variables	Only non-static declarations + final static fields
Anonymous Class	As expression in non-static context	None	Yes	All members in enclosing context + final local variables	Only non-static declarations + final static fields
	As expression in static context	None	No	Static members in enclosing context + final local variables	Only non-static declarations + final static fields

Assignment

1. What do you mean by class and object? How can you use class and object in java?
2. What do you mean by constructor?
3. What is method overloading and overriding?
4. Explain various access specifier?
5. Explain the concept of inheritance with suitable example?
6. What do you mean by package? What are the uses of it?
7. What is abstract and final classes and methods?
8. What do you mean by static members?
9. What do you mean by interface? Compare and contrast with class.
10. Describe this and super methods?
11. Explain various nested classes.