

Object Oriented Programming 1

Lecture 11: Java Packages / API, Java Inheritance & Java Polymorphism

By

Elubu Joseph

MSci.IS

Email: josebulinda@gmail.com

or

jose@kumiuniversity.ac.ug

Agenda

1. Java Packages / API,
2. Java Inheritance &
3. Java Polymorphism

Java Packages / API,

Java Package

Package is a collection of related classes. Java uses package to group related classes, interfaces and sub-packages in any Java project.

We can assume package is a folder or a directory that is used to store similar files.

Using packages:-

1. avoid name conflicts
2. control access of class, interface and enumeration etc.
3. Makes it easier to locate the related classes
4. provides a good structure for projects with hundreds of classes and other files.

Types of Java Package

Package can be built-in and user-defined, Java provides rich set of built-in packages in form of API that stores related classes and sub-packages.

1. **Built-in Package:** javax, sql, math, util, lang, i/o etc are the example of built-in packages.
2. **User-defined-package:** Java package created by user to categorize their project's classes and interface are known as user-defined packages.

How to Create a Package

Creating a package in java is quite easy, simply include a package command followed by name of the package as the first statement in java source file.

```
package lecture20;  
public class employee{  
    String empId;  
    String name;  
}
```

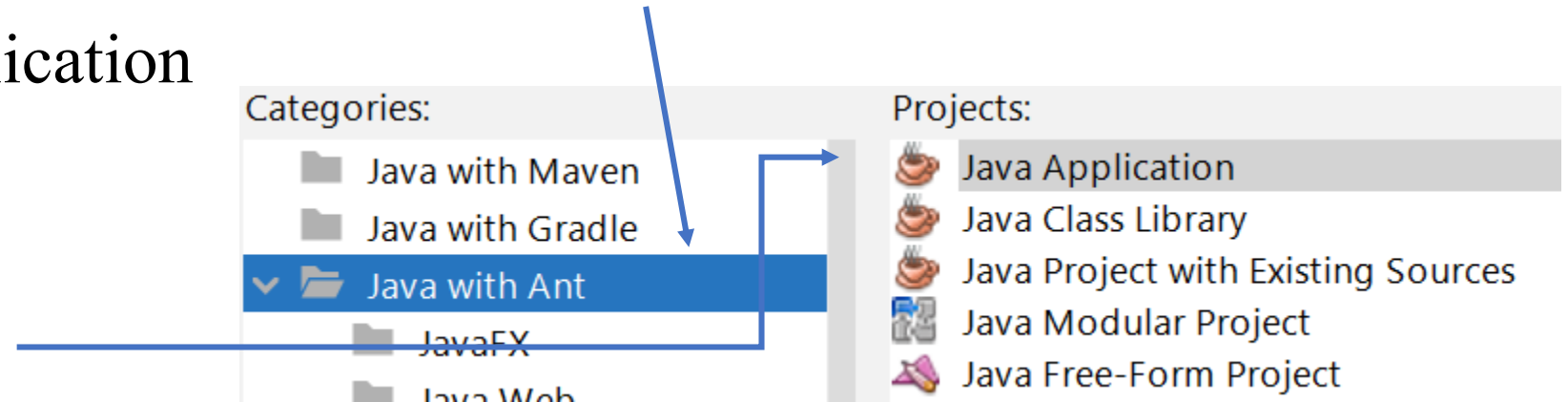
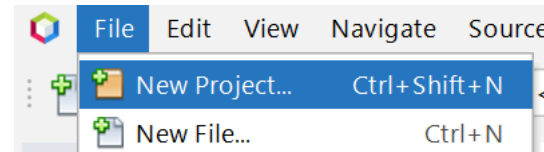
The above statement will create a package name **lecture20** where employee.java file will be stored.

How to Create a Package using Netbeans IDE

We can also create packages through our IDEs after creating a project. Let's create a package called Legged inside the project called Human.

First. Let's create a project above.

1. Click on File>New Project
2. Select project category e.g. Java with Ant
3. Click Java Application
- 4 Click Next



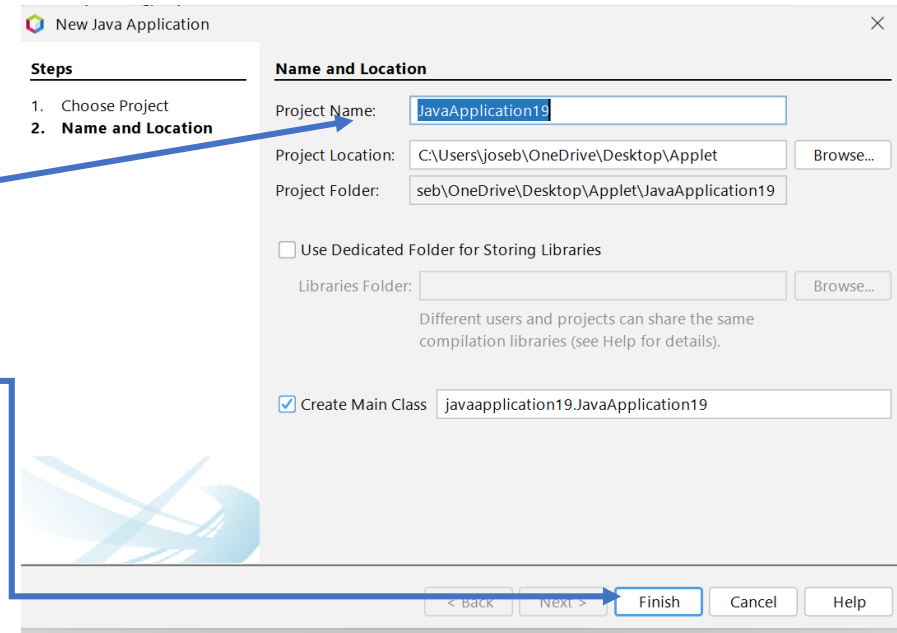
How to Create a Package using Netbeans IDE

We can also create packages through our IDEs after creating a project. Let's create a package called Legged inside the project called Human.

First. Let's create a project above.

5. Enter Project name

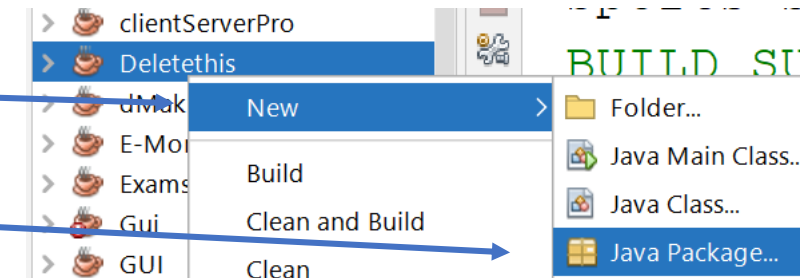
6. Click Finish



Creating Package using Netbeans IDE

After creating the project, you can now create a package. To create a package,

1. Right click on the project you just created
2. Point at New
3. Click Java Package
4. Enter the name of the package e.g. lecture20
5. Click Finish



Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Package Name:

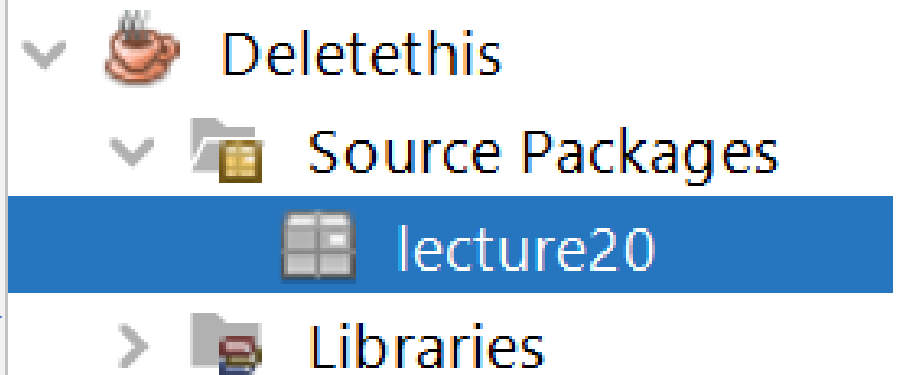
Project:

Location:

Created Folder:

< Back Next > **Finish** Cancel Help

When you see this display, then you are done



Additional points about package

1. Package statement must be first statement in the program even before the import statement.
2. A package is always defined as a separate folder having the same name as the package name.
3. Store all the classes in that package folder.
4. All classes of the package which we wish to access outside the package must be declared public.
5. All classes within the package must have the package statement as its first line.
6. All classes of the package must be compiled before use.

How to import Java Package

To import java package into a class, we need to use java **import** keyword which is used to access package and its classes into the java program.

Use import to access built-in and user-defined packages into your java source file so that your class can refer to a class that is in another package by directly using its name.

There are 3 different ways to refer to any class that is present in a different package:

How to import Java Package +

There are 3 different ways to refer to any class that is present in a different package:

1. without import keyword
2. import package with specified class
3. import package with all classes

Accessing package without import keyword

If you use fully qualified name to import any class into your program, then only that particular class of the package will be accessible in your program, other classes in the same package will not be accessible.

With this approach, there is no need to use the `import` statement. But you will have to use the fully qualified name every time you are accessing the class or the interface.

This is generally used when two packages have classes with same names. For

Example: `java.util` and `java.sql` packages contain `Date` class.

Accessing package without import keyword+

Example

In this example, we are creating a class called A1 in package lecture10 and class called B1 in package lecture10_2, we are accessing it while creating its object of class A.

```
package lecture10;  
public class A1 {  
    public void msg() {  
        System.out.println("Using fully qualified  
names");  
    }  
}
```

Accessing package without import keyword+

```
package lecture10_2;
class B1{
    public static void main(String args[]) {
        lecture10.A1 ob = new lecture10.A1(); //using
        fully qualified name
        ob.msg();
    }
}
```

Output: Using fully qualified names

Note lecture10.A1 in the creation of A1 class object.

Import the Specific Class

Package can have many classes but sometimes we want to access only specific class in our program in that case, Java allows us to specify class name along with package name.

If we use import `packagename.classname` statement then only the class with name classname in the package will be available for use.

Import the Specific Class- A1 class

```
package lecture10;  
public class A1 {  
    public void msg() {  
        System.out.println("Using fully qualified  
names");  
    }  
}
```

Import the Specific Class- B1 class

```
package lecture10_2;
import lecture10.A1;
class B1 {
    public static void main(String args[]) {
        A1 ob = new A1();
        ob.msg();
    }
}
```

Output: Using fully qualified names

Import all classes of the package

If we use **packagename.*;** statement, then all the classes and interfaces of this package will be accessible but the classes and interface inside the sub-packages will not be available for use.

The **import** keyword is used to make the classes of another package accessible to the current package.

Example .

In this example, we created a class First in **lecuture10** package that access it in another class Second by using import keyword.

Import all classes of the package

First class

```
package lecture10;  
public class First {  
    public void msg() {  
        System.out.println("Using fully qualified  
names");  
    }  
}
```

Import all classes of the package

Second class

```
package lecture10_2;
import lecture10.*;
class Second {
    public static void main(String args[]) {
        First ob = new First();
        ob.msg();
    }
}
```

Output: Using fully qualified names

Java Inheritance &

Inheritance (IS-A relationship)

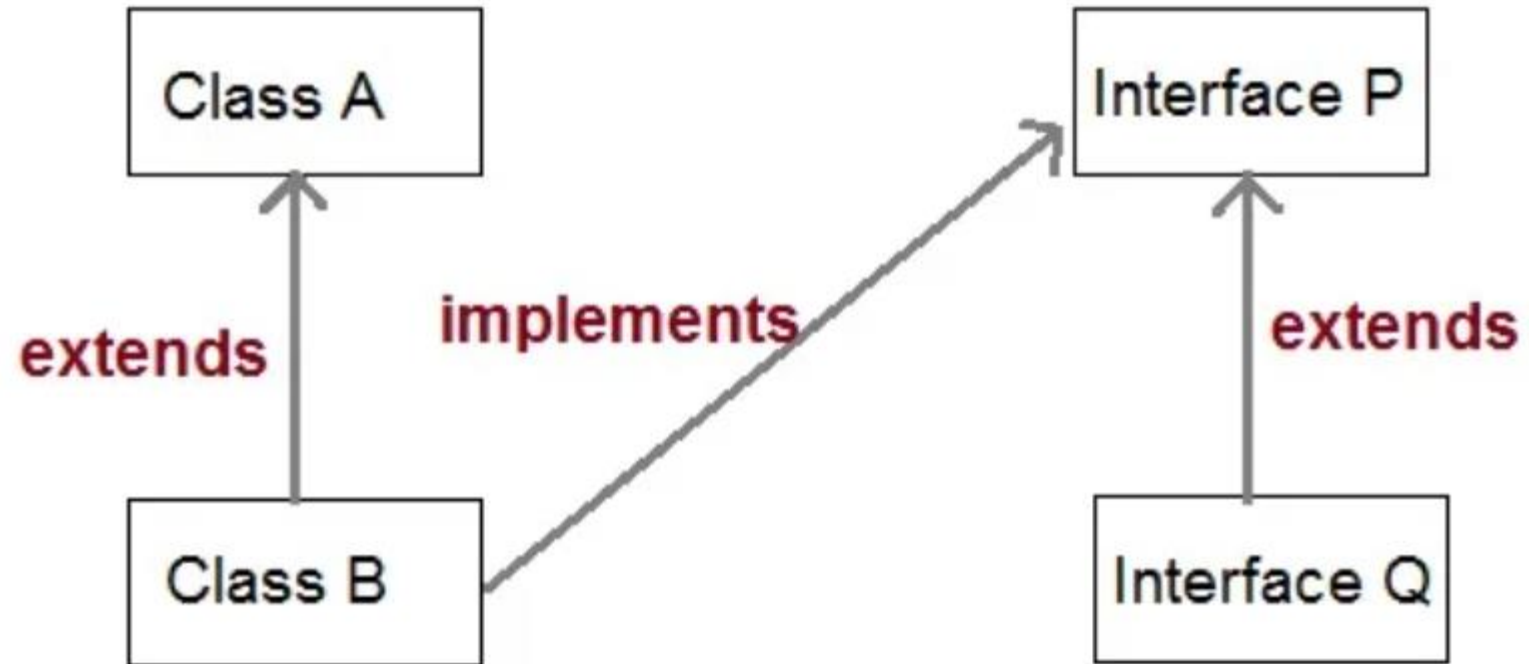
Inheritance is OOP features that provided a mechanism which allow **a class to inherit property of another class**. When a Class extends another class it inherits all **non-private** members including fields and methods.

Inheritance in Java can be best understood in terms of Parent and Child relationship, also known as **Super class**(Parent) and **Sub class**(child) in Java language.

Inheritance (IS-A relationship)

Inheritance defines **is-a** relationship between a Super class and its Sub class. **extends** and **implements** keywords are used to describe inheritance in Java.

In Java, we have two types of relationship: **Is-A** relationship: Whenever one class inherits another class, it is called an IS-A relationship. **Has-A** relationship: Whenever an instance of one class is used in another class, it is called HAS-A relationship.



Using **extends** Keyword

Let us see how **extends** keyword is used to achieve Inheritance. It shows super class and sub-class relationship.

```
class Human { ..... }  
class Man extends Human {  
    ..... //extends the property of  
vehicle class  
}
```

Now based on above example. In OOPs term we can say that, **Human** is super class of **Man** while **Man** is sub class of **Human**. Hence achieving Man IS-A Human relationship.

Simple example of Inheritance

Before moving ahead let's take a quick example and try to understand the concept of Inheritance better.

In this example we will create two classes i.e Parent class(super) and Child class(sub class), all located in lecture11 package. This is to help us understand inheritance.

Parent Class

```
package lecture11;  
class Parent {  
    public void p1() {  
        System.out.println("Parent method");  
    }  
}
```

Child class

```
package lecture11;

public class Child extends Parent{
    public void c1() {
        System.out.println("Child method");
    }

    public static void main(String[] args) {

        Child cob = new Child();
        cob.c1(); //method of Child class
        cob.p1(); //child object accessing method of Parent class
    }
}
```

OUTPUT
Child method
Parent method

Note.

In the code above we have a class **Parent** which has a method **p1()**. We then created a new class **Child** which inherits the class **Parent** using the **extends** keyword and defines its own method **c1()**.

Now by virtue of inheritance the class **Child** can also access the **public** method **p1()** of the class **Parent**.

Inheriting variables of super class

All the members of super class implicitly inherits to the child class.
Member consists of instance variable and methods of the class.

Example

In this example the sub-class will be accessing the variable defined in the super class.

Vehicle Class-Super

```
class Vehicle {  
    // variable defined  
    String vehicleType;  
  
}
```

Car class - sub

```
public class Car extends Vehicle {  
    String modelType;  
    public void showDetail() {  
        vehicleType = "Subaru"; //accessing Vehicle class member  
variable  
        modelType = "Sports";  
        System.out.println(modelType + " " + vehicleType);  
    }  
    public static void main(String[] args) {  
        Car ob = new Car();  
        ob.showDetail();  
  
    }  
}
```

OUTPUT Sports Subaru

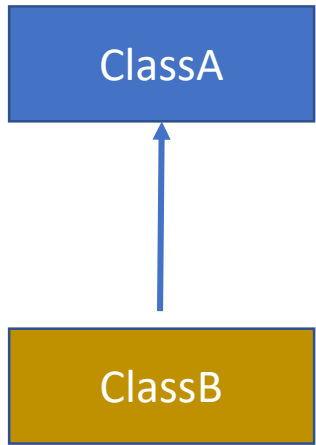
Types of Inheritance

Java mainly supports only three types of inheritance that are listed below.

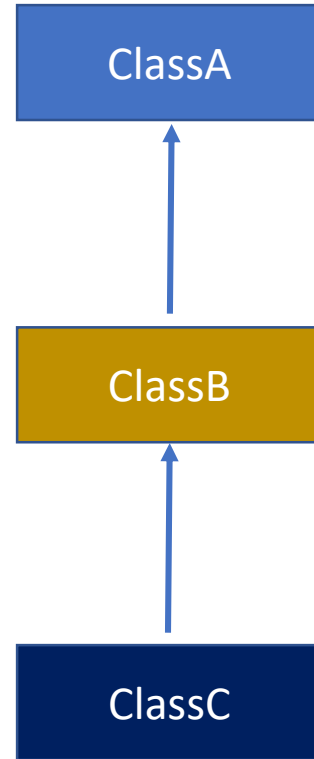
1. Single Inheritance
2. Multilevel Inheritance
3. Heirarchical Inheritance

NOTE: Multiple inheritance is not supported for classes but allowed for interfaces in java. We can get a quick view of type of inheritance from the image below.

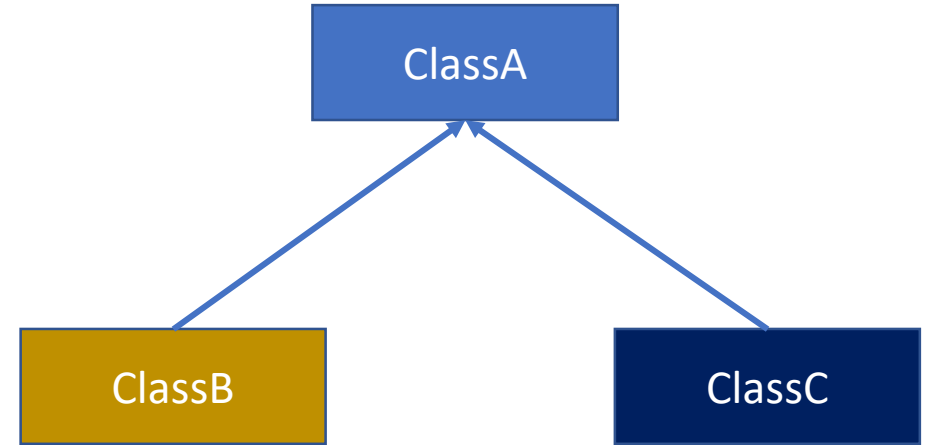
Types of inheritance



1. Single Inheritance



2. Multilevel Inheritance



3. Hierarchical Inheritance

Single Inheritance

When a class extends to another class then it forms single inheritance. In the example below, we have two classes in which class A extends to class B that forms single inheritance.

```
class A{  
    int a = 60;  
    void show(){  
        System.out.println("A = "+a);  
    }  
}
```

```
public class B extends A{  
    public static void main(String[] args) {  
        B b = new B();  
        b.show(); //child can access parent method show()  
    }  
}
```

What will be the output here?

Single Inheritance program explained

Here, we can notice that `show()` method is declared in class A, but using child class B b, we can call it. That shows the inheritance between these two classes.

Multilevel Inheritance

When a class extends to another class that also extends some other class forms a multilevel inheritance. For example a class C extends to class B that also extends to class A and all the data members and methods of class A and B are now accessible in class C.

Example.

```
class A{  
    int a = 60;  
    void show(){  
        System.out.println("A = "+a);  
    }  
}
```

```
class B extends A{  
    int b = 20;  
    void showB() {  
        System.out.println("B = "+b);  
    }  
}
```

Class C access Class A through B

```
public class C extends B{  
    public static void main(String[] args) {  
        C c = new C();  
        c.show();  
        c.showB();  
    }  
}
```

OUTPUT:

A = 60

B = 20

Hierarchical Inheritance

When a class is extended by two or more classes, it forms hierarchical inheritance.

For example, class B extends to class A and class C also extends to class A in that case both B and C share properties of class A.

```
class A{  
    int a = 60;  
    void show(){  
        System.out.println("A = "+a);  
    }  
}
```

```
class B extends A{  
    int b = 20;  
    void showB() {  
        System.out.println("B = "+b);  
    }  
}
```

Class C and Class B access A while in C.

```
public class C extends B{  
    public static void main(String[] args) {  
        C c = new C();  
        c.show();  
        B b = new B();  
        b.show();  
    }  
}
```

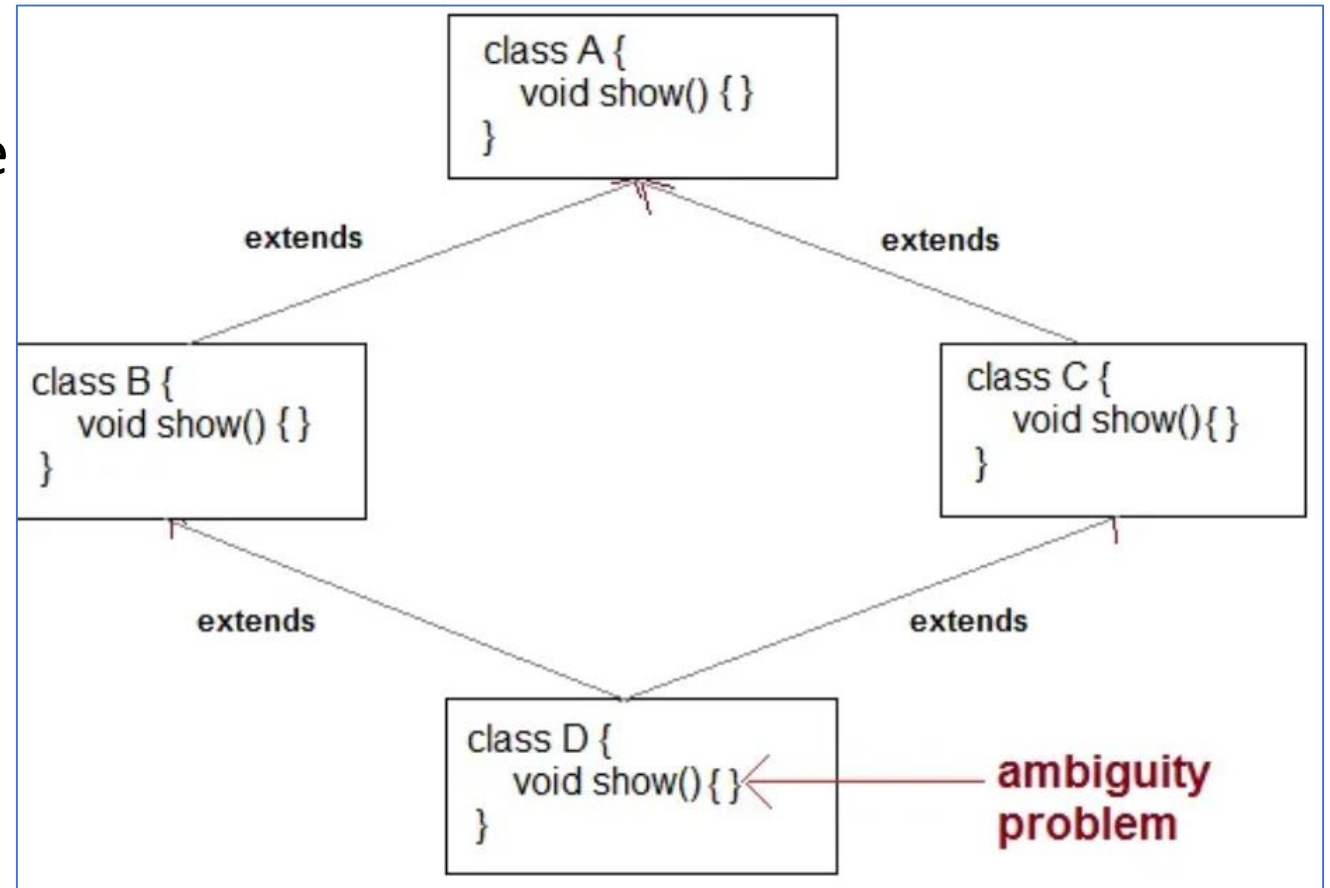
OUTPUT:

A = 60

B = 60

Why multiple inheritance is not supported in Java classes?

1. To remove ambiguity.
2. To provide more maintainable and clear design.

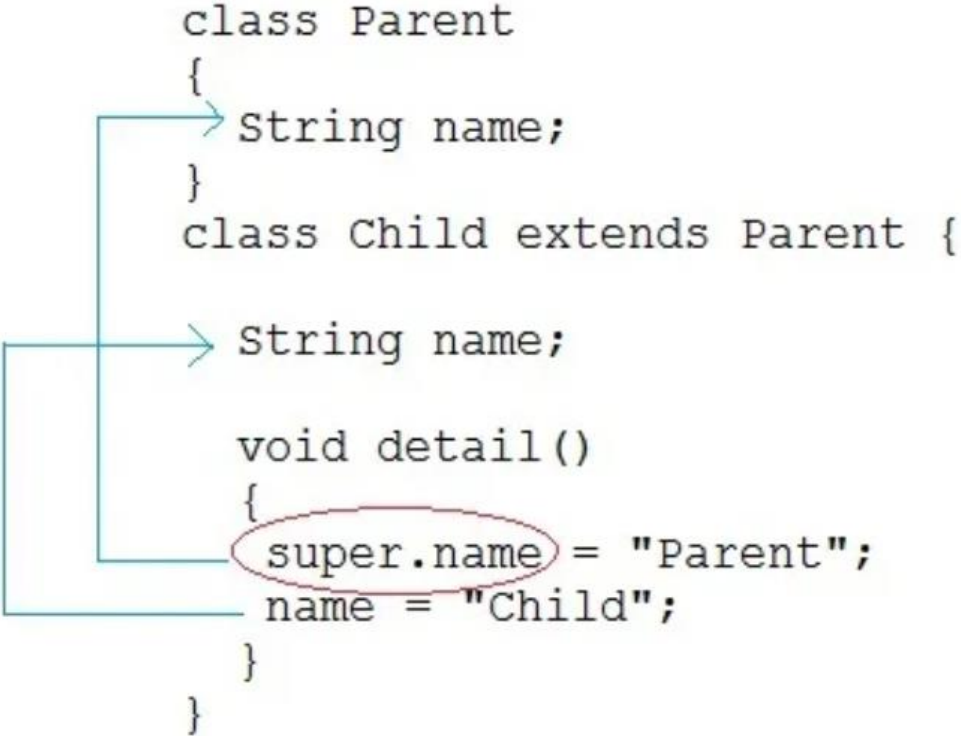


super keyword

In Java, **super** keyword is used to refer to immediate parent class of a child class. In other words **super** keyword is used by a subclass whenever it need to refer to its immediate super class.

```
class Parent
{
    String name;
}
class Child extends Parent {
    String name;

    void detail()
    {
        super.name = "Parent";
        name = "Child";
    }
}
```



Example of Child class referring to Parent class property
using **super** keyword

In this example we will only focus on accessing the parent class property or variables.

Parent1 class

```
class Parent1{  
String name;  
}
```

Child1 class uses super keyword

```
public class Child1 extends Parent1{
    String name;
    public void details(){
        super.name = "Parent"; //refers to parent class
member
        name = "Child";
        System.out.println(super.name+" and "+name);
    }
    public static void main(String[] args) {
        Child ob = new Child();
        ob.details();
    }
}
```

Output

Parent and Child

Example of Child class calling Parent class `constructor` using `super` keyword

In this example we will focus on accessing the parent class constructor

Parent Class with a constructor

```
class Parent2 {  
    String name;  
    public Parent2 (String n) {  
        name = n;  
    }  
}
```

Child2 class access Parent2 class constructor using super keyword

```
public class Child2 extends Parent2{
    String name;
    public Child2(String n1, String n2) {
        super(n1); //passing argument to parent class constructor
        this.name = n2;
    }
    public void details() {
        System.out.println(super.name+" and "+name);
    }
    public static void main(String[] args) {
        Child2 cobj = new Child2("Parent","Child member");
        cobj.details();
    }
}
```

Output: Parent and Child member

Note:

When calling the parent class constructor from the child class using super keyword, super keyword should always be the first line in the method/constructor of the child class.

Qn. Can you use both `this()` and `super()` in a Constructor?

Yes, because both `super()` and `this()` can be used as statement inside a constructor.

Purpose of Inheritance

It promotes the code reusability i.e the same methods and variables which are defined in a parent/super/base class can be used in the child/sub/derived class.

It promotes polymorphism by allowing method overriding.

Disadvantages of Inheritance

Main disadvantage of using inheritance is that the two classes (parent and child class) gets **tightly coupled** that means any change made in super class affect sub classes as well.

Java Polymorphism

Java Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; **Inheritance** lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a superclass called `Animal` that has a method called `animalSound()`. Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

Example



```
class Animal {  
    public void animalSound() {  
        System.out.println("The animal makes a sound");  
    }  
}
```

```
class Pig extends Animal {  
    public void animalSound() {  
        System.out.println("The pig says: wee wee");  
    }  
}
```

```
class Dog extends Animal {  
    public void animalSound() {  
        System.out.println("The dog says: bow wow");  
    }  
}
```


Remember from the Inheritance chapter that we use the **extends** keyword to inherit from a class.

Now we can create **Pig** and **Dog** objects and call the **animalSound()** method on both of them:

Example

```
class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}
```

```
class AnimalRunner {  
    public static void main(String[] args) {  
        Animal myAnimal = new Animal(); // Create a Animal object  
        Pig myPig = new Pig(); // Create a Pig object  
        Dog myDog = new Dog(); // Create a Dog object  
  
        myAnimal.animalSound();  
        myPig.animalSound();  
        myDog.animalSound();  
    }  
}
```

The animal makes a sound

Pigs say: wee wee

The Dog Say: bow bow

OUTPUT

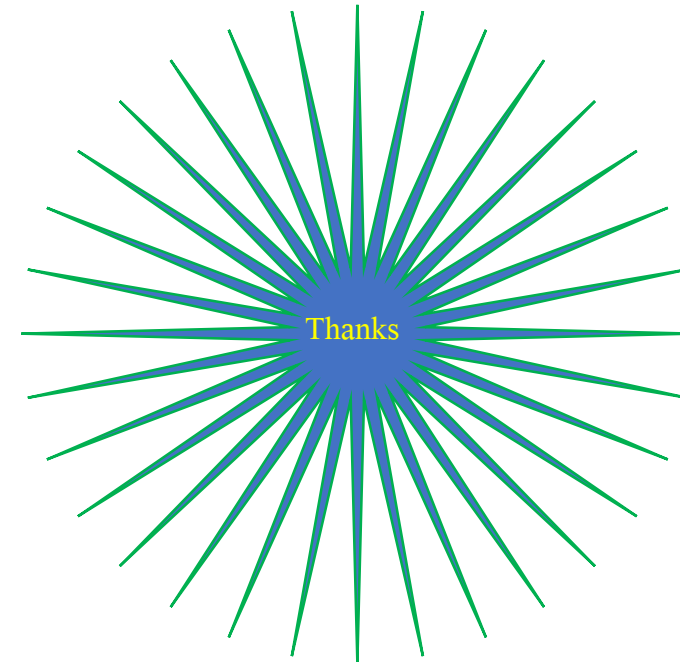
Why And When To Use "Inheritance" and "Polymorphism"?

- It is useful for code reusability: reuse attributes and methods of an existing class when you create a new class.

Summary

1. Java Packages / API, (How to create packages, importance of using packages etc.
2. Java Inheritance (types of inheritance, how to inherit/implement why it is important etc.)
3. Java Polymorphism(Much on overriding etc.)

Thank you for
Listening



Reference

What is is-A-relationship in java? GeeksforGeeks. (2021, December 1). Retrieved June 2, 2022, from <https://www.geeksforgeeks.org/what-is-is-a-relationship-in-java/#:~:text=In%20Java%2C%20we%20have%20two,is%20called%20HAS%2DA%20relationship.>

Inheritance (IS-a relationship) in Java. Studytonight.com. (n.d.). Retrieved June 2, 2022, from <https://www.studytonight.com/java/inheritance-in-java.php>

Java package. Studytonight.com. (n.d.). Retrieved June 2, 2022, from <https://www.studytonight.com/java/package-in-java.php>