DePaul University

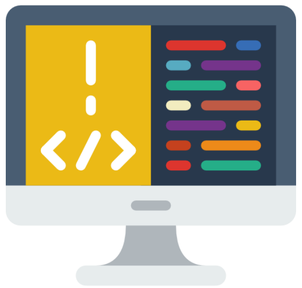
OOP Principles:

Polymorphism

Object-oriented Software Development
SE 350– Spring 2021

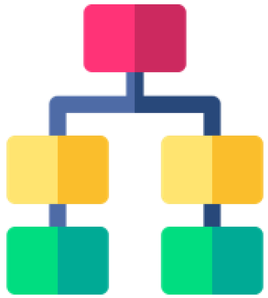
Vahid Alizadeh

Week 3.2
April 15, 2021



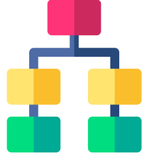
Object-oriented Programming

Principles



Object-oriented Programming Principles

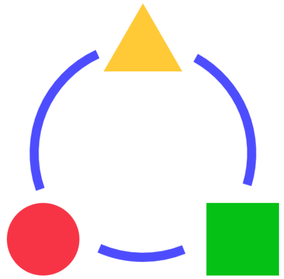
INHERITANCE



Inheritance Constructor Chaining Example

- **Example** [package oopPrinciples.inheritance.discussion]
- **Parent class constructor is always called**
- **You can't have both this() and super() in the same constructor.**
  - To avoid multiple super calls in the calls of constructor chaining.
- **Always maintain the proper constructor chaining.**
- **Summary…**

```
class ParentClsChain {
    int i;
    ParentClsChain() {
        System.out.println("Invoking parameterless constructor of Parent class.");
    }
}
class ChildClsChain extends ParentClsChain {
    int b;
    ChildClsChain() {
//      both this() and super() cannot be used together
//        super();
        this(450);
        System.out.println("Invoking parameterless constructor of Child.");
    }
    ChildClsChain(int b) {
        this.b = b;
        System.out.println("Inside Child constructor with one parameter where b= " + b);
    }
}
class DemoConstructorChaining {
    public static void main(String[] args) {
        System.out.println("***Analysis of this() and super() keyword.***\n");
        ChildClsChain sampleObj = new ChildClsChain();
    }
}
```



Object-oriented Programming Principles

POLYMORPHISM

Compile-time Polymorphism

- **Polymorphism = "*one name with many forms*"**

1. Compile-time polymorphism

- The compiler can bind the methods to the respective objects at compile time.
- = Static binding or Early binding
- Method overloading
  - Method parameters can vary with a number, order, or the types of parameter

- **Example** [package oopPrinciples.polymorphism]
  - Method names are the same, Method signature are different.

```
package oopPrinciples.polymorphism;

//Method overloading example
public class CompileTimePolymorphism {
}

class Addition {
    int sum(int x, int y) {
        return x + y;
    }
    double sum(double x, double y) {
        return x + y;
    }
    String sum(String s1, String s2) {
        return s1.concat(s2);
    }
}
class DemoCompileTimePolymorphism {
    public static void main(String[] args) {
        System.out.println("***Method overloading example***");
        Addition additionObj = new Addition();
        int sumOfIntergers = additionObj.sum(400, 50);
        System.out.println("400 + 50 is :" + sumOfIntergers);
        double sumOfDoubles = additionObj.sum(399.5, 50.5);
        System.out.println("399.5 + 50.5 is :" + sumOfDoubles);
        String sumOfStrings = additionObj.sum("Vahid", "Alizadeh");
        System.out.println("'Vahid'+ 'Alizadeh' is :" + sumOfStrings);
    }
}
```

Method Overloading Discussion

- **A class cannot have multiple methods with the same signature and different primitive return types.**
- **What is method signature?**
  - Method name and its parameters(number, type, order)
- **Question: Which code snippet is a method overloading example?**
- **Can we have constructor overloading?** Yes

```
class Addition
{
  public int sum(int x, int y) {
        return x + y;
    }
  public double sum(int x, int y, int z) {
        return x + y+ z;
    }
}
```

1

```
class Addition
{
  public int sum(int x, int y) {
    return x + y;
  }
  public double sum(int x, int y) {
    return x + y;
  }
}
```

2

Run-time Polymorphism

2. Runtime Polymorphism

- Call to an overridden method is resolved dynamically at runtime
- Method overriding
- = Dynamic binding = Dynamic method dispatch

```
public class Animal {
    public void makeNoise()
    {
        System.out.println("Some sound");
    }
}

class Dog extends Animal{
    public void makeNoise()
    {
        System.out.println("Bark");
    }
}

class Cat extends Animal{
    public void makeNoise()
    {
        System.out.println("Meawoo");
    }
}
```

```
public class Demo
{
    public static void main(String[] args) {
        Animal a1 = new Cat();
        a1.makeNoise(); //Prints Meowoo

        Animal a2 = new Dog();
        a2.makeNoise(); //Prints Bark
    }
}
```

Method Overriding Discussion

- **Example** [package oopPrinciples.polymorphism]
  - Overridden method (in parent) vs. Overriding method (in child)
- **Key benefit of method overriding?**
  - Can change the behavior of the parent class
- **In method overriding, the return type matters.**
  - Method's return type of the parent and child must be compatible.

```
//Method overriding example
public class RuntimePolymorphism {
}
class ParentClass {
    // Overridden method
    public void showMe( ) {
        System.out.println("Inside ParentClass.showMe( )");
    }
    public void doNotChangeMe( ) {
        System.out.println("Inside ParentClass.doNotChangeMe( ).");
    }
}
class ChildClass extends ParentClass {
    // Overriding method
    public void showMe( ) {
        System.out.println("Inside ChildClass.showMe( ).");
    }
}
class DemoRuntimePolymorphism {
    public static void main(String[] args) {
        System.out.println("*** Method overriding example.***");
        ChildClass childOb = new ChildClass( );
        childOb.doNotChangeMe( );
        childOb.showMe( );//Will display the overridden method
    }
}
```

Method Overriding Discussion (2)

- **Runtime polymorphism doesn't happen for static methods.**
- **Example** [package oopPrinciples.polymorphism.discussion]

  - Calling a static method >> executes the method in the reference type
    - ✕ NO Polymorphism
  - Calling a dynamic method >> executes the method in the Actual object.
    - ✔ Polymorphism

  - Static methods always refer to the reference type.
  - Calling a static method means you want to execute the method in the reference type.

```java
package oopPrinciples.polymorphism.discussion;

//Runtime polymorphism doesn't happen for static methods.
public class OverridingStaticMethod {
}

class ParentClass {
    public static void printStatic(){
        System.out.println("static from ParentClass");
    }
    public void printDynamic(){
        System.out.println("dynamic from ParentClass");
    }
}

class ChildClass extends ParentClass {
    public static void printStatic(){
        System.out.println("static from ChildClass");
    }
    public void printDynamic(){
        System.out.println("dynamic from ChildClass");
    }
}

class DemoOverridingStaticMethod{
    public static void main (String args[]) {
        ParentClass parentClass = new ChildClass();
        parentClass.printStatic();
        parentClass.printDynamic();
        ChildClass childClass = new ChildClass();
        childClass.printStatic();
        childClass.printDynamic();
    }
}
```

Upcasting vs. Downcasting

- **Casting types:**
  - Upcasting
    - Casting to super type
  - Downcasting
    - Casting to the sub type
- **Example** [package oopPrinciples.polymorphism]
- **Downcasting is risky**
  - can cause a runtime exception

```java
//Example of Upcasting and Downcasting
public class UpDownCasting {
}
class Vehicle {
    public void showMe( ) {
        System.out.println("Inside Vehicle.showMe( )");
    }
}
class Bus extends Vehicle {
    public void showMe( ) {
        System.out.println("Inside Bus.showMe( )");
    }
    public void specificMethod( ) {
        System.out.println("Inside Bus.showMe( )");
    }
}
class Train extends Vehicle {
    public void showMe( ) {
        System.out.println("Inside Train.showMe( )");
    }
    public void specificMethod( ) {
        System.out.println("Inside Train.specificMethod( )");
    }
}
class DemoUpDownCasting {
    public static void main(String[] args) {
        System.out.println("***Demo Upcasting and Downcasting.***\n");
//      ####Upcasting
        Vehicle obVehicle=new Train();//ok
        obVehicle.showMe();//Output: Inside Train.showMe( )
//      obVehicle.specificMethod();//error //Since the apparent type in the code is a Vehicle,
//      but not a Train. Need downcasting Line43 and 44

//      Creating two subtype(one Bus and one Train) object
        Bus obBus=new Bus();
        Train obTrain=new Train();

//      ####Downcasting :Casting to a subtype
//      obBus=(Bus)obVehicle;       //Run-time error:Train cannot be cast to Bus //If you uncomment
this line, you will not receive any compilation errors, but you will
//      encounter a runtime exception

        obTrain=(Train)obVehicle;   //Ok, this time it is ok.
        obTrain.specificMethod();   //also ok //Output: Inside Train.specificMethod( )
    }
}
```

Casting Discussion

- **You can refer to a child class object via a parent class reference, but the reverse is not true.**
  - Why? Compare with real-life examples: Vehicle/Bus or Shape/Rectangle
  - Always do an “is-a” test
- **An object can:**
  - **Implicitly upcast** to a super class reference
  - **Explicitly downcast** to a derived class reference
- **Why the following code will be resolved at runtime?**

```
Vehicle obVehicle = new Bus();
obVehicle.showMe();
```

  - Check this Example
    - [package oopPrinciples.polymorphism.discussion]

```
import java.util.Random;

//Demonstration of why a code will be resolved at runtime
public class RuntimeExample {
}

class Vehicle {
    public void showMe() {
        System.out.println("Inside Vehicle.showMe()");
    }
}
class Bus extends Vehicle {
    public void showMe() {
        System.out.println("Inside Bus.showMe()");
    }
}
class Taxi extends Vehicle {
    public void showMe() {
        System.out.println("Inside Taxi.showMe()");
    }
}
class DemoRunTimeDiscussion {
    public static void main(String[] args) {
        System.out.println("***Demo analysing runtime polymorphism ***\n");
        Vehicle obVehicle;
        int count = 0;
        Random random = new Random();
//      Considering 5 choices
        while (count < 5) {
            int tick = random.nextInt(10);//0 to 9
            if (tick % 2 == 0) {
                obVehicle = new Bus();
            } else {
                obVehicle = new Taxi();
            }
            obVehicle.showMe();// Output will be determined at runtime
            count++;
        }
    }
}
```

Final Keyword

- *Final* **Keyword**
  - prevent the inheritance process
  - preserve the consistent state of the object
  - Final Class: Immutable class and prevent inheritance
    - (Also, private constructors)
  - Final Method: Prevent method overriding
  - Final Variable: Constant variable
  - Final Constructor? NO!
    - “Because constructor declarations are not members. They are never inherited and therefore are not subject to hiding or overriding”

```java
//The final keyword used to prevent inheritance
//Compilation error: cannot subclass the final class
final class ParentClass {
  public void showMe(){
    System.out.println("Inside Parent.showMe()");
  }
}
class ChildClass extends ParentClassTest4
{
    //Some code
}
```

```java
//The final keyword used to prevent method overriding
//Compilation error: Cannot override the final method
class ParentClass {
  final public void showMe(){
    System.out.println("Inside Parent.showMe()");
  }
}
class ChildClass extends ParentClassTest4
{
    public void showMe() { // error
        System.out.println("Inside Parent.showMe()");
    }
}
```

```java
final double PI=3.14
```

Method Overriding Discussion (3)

Overriding private methods

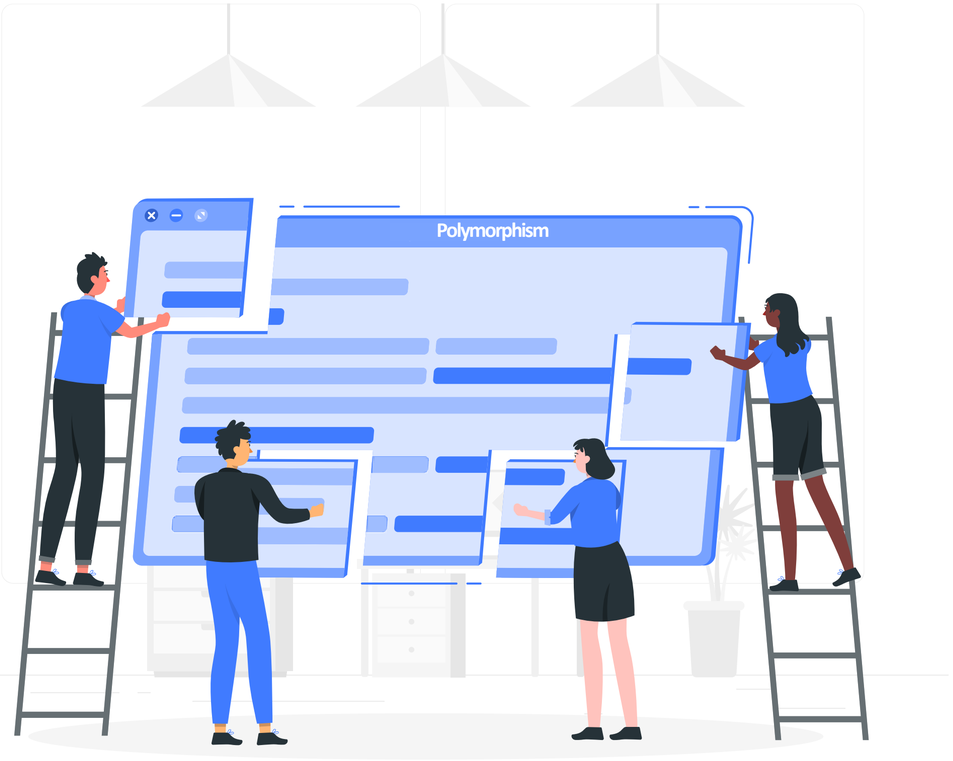
- ❌ **It is not possible to override a private methods.**
- **Example** [package oopPrinciples.polymorphism.discussion]
  - Compile time error .
  - The error tries to call the parent method.

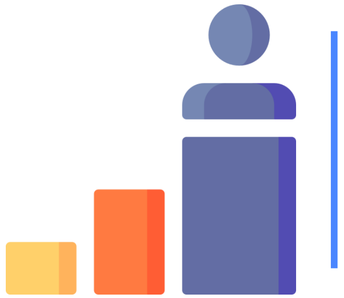
```java
package oopPrinciples.polymorphism.discussion;

public class OverridingPrivateMethod {
}
//It is not possible to override a private methods.
class Parent {
    private void printSomething() {
        System.out.println("Parent Method");
    }
}
class Child extends Parent {
    private void PrintSomething() {
        System.out.println("Child Method");
    }
}
class DemoOverridingPrivateMethod{
    public static void main(String[] args) {
        Parent obj = new Child();
        //Error: PrintSomething() has private access in parent (not the child)
        obj.printSomething();
    }
}
```

Summary: Polymorphism

- **Why should you write polymorphic code?**
  - Flexible code
  - Adopt upcoming changes
- **How inheritance and polymorphism are related?**
- **What is difference between inheritance and polymorphism?**
- **Summary**
  - Covered topics





Class Activity

Get Ready!

Class Activity Week 3.2

Get ready to compete!

Output of following Java Program?

Output of following Java Program?

```java
class Parent {
    public void show() {
       System.out.println("Parent::show() called");
    }
}

class Child extends Parent {
    public void show() {
       System.out.println("Child::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Parent p = new Child();;
        p.show();
    }
}
```

Child::show() called

Parent::show() called

None of the above

Total Results: 16

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Output of following Java Program?

```
class Parent {
    public void show( ) {
       System.out.println("Parent::show( ) called");
    }
}

class Child extends Parent {
    public void show( ) {
       System.out.println("Child::show( ) called");
    }
}

public class Main {
    public static void main(String[] args) {
        Parent p = new Child( );;
        p.show( );
    }
}
```

Child::show() called

Parent::show() called

None of the above

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Leaderboard

1	A	alex	3972
2	M	Michael Allen	3903
3	N	Nathan	3861
4	S	Steve	3750
5	N	Nathaniel Waldschmidt	3000
5	A	alzlaz	3000
7	S	Syed	2805
8	S	Saiyed	2778
9	N	NP	2764
	M	Michael Kotwica	

Output of following Java Program?

Output of following Java Program?

```java
class Parent {
    final public void show() {
       System.out.println("Parent::show() called");
    }
}

class Child extends Parent {
    public void show() {
       System.out.println("Child::show() called");
    }
}

class Main {
    public static void main(String[] args) {
        Parent p = new Child();;
        p.show();
    }
}
```

- Parent::show() called
- Child::show() called
- Compiler Error
- Runtime Error

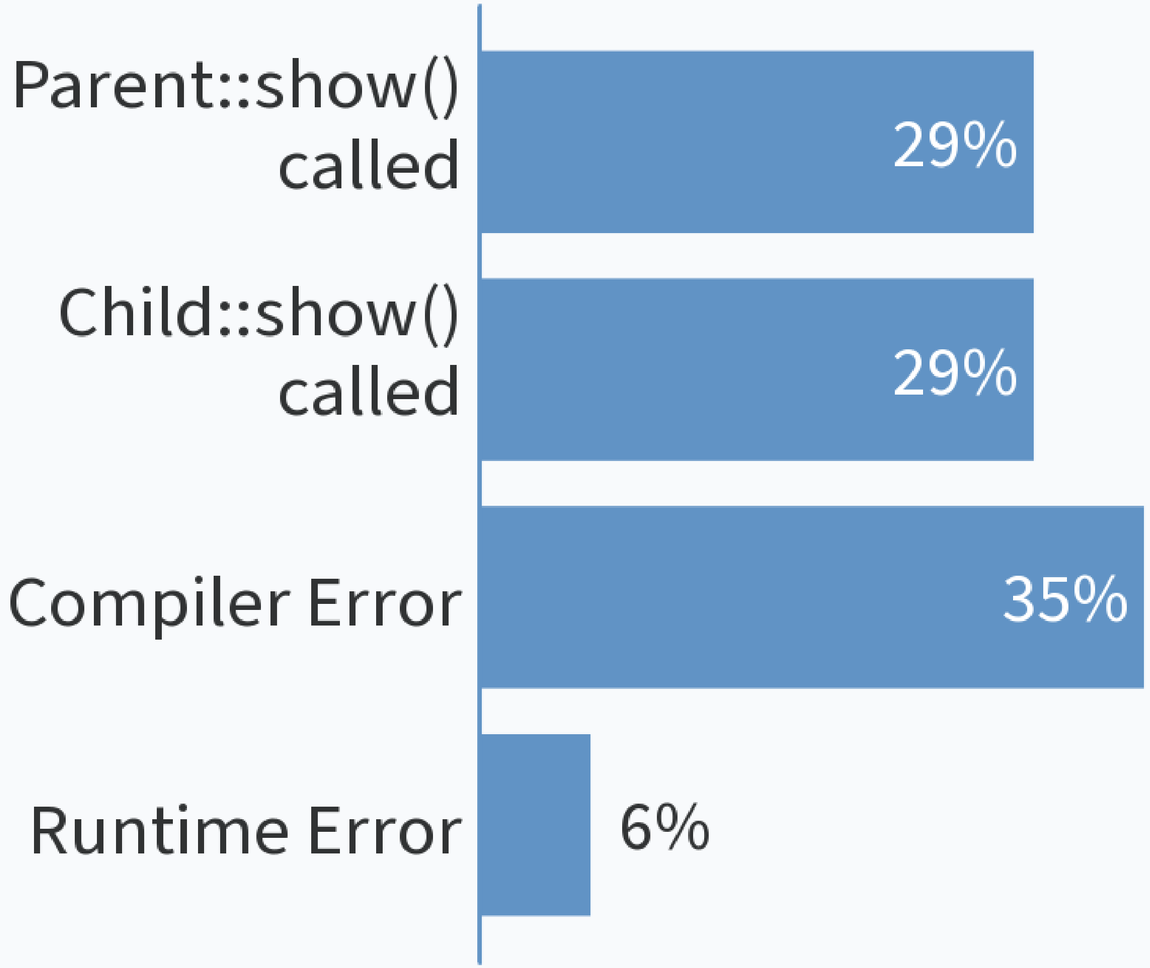
Total Results: 0

Output of following Java Program?

```
class Parent {
    final public void show() {
       System.out.println("Parent::show() called");
    }
}

class Child extends Parent {
    public void show() {
       System.out.println("Child::show() called");
    }
}

class Main {
    public static void main(String[] args) {
        Parent p = new Child();;
        p.show();
    }
}
```



Leaderboard

1	A	alex	3972
2	M	Michael Allen	3903
3	N	Nathan	3861
4	S	Steve	3750
5	N	Nathaniel Waldschmidt	3000
5	A	alzlaz	3000
7	S	Syed	2805
8	S	Saiyed	2778
9	N	NP	2764
	M	Michael Kotwica	

Output of following Java Program?

Output of following Java Program?

```
class Parent {
    public static void show() {
       System.out.println("Parent::show() called");
    }
}

class Child extends Parent {
    public static void show() {
       System.out.println("Child::show() called");
    }
}

class Main {
    public static void main(String[] args) {
        Parent p = new Child();;
        p.show();
    }
}
```

Parent::show() called

Child::show() called

Compiler Error

Total Results: 0

Output of following Java Program?

```
class Parent {
    public static void show() {
       System.out.println("Parent::show() called");
    }
}

class Child extends Parent {
    public static void show() {
       System.out.println("Child::show() called");
    }
}

class Main {
    public static void main(String[] args) {
        Parent p = new Child();;
        p.show();
    }
}
```

Parent::show() called

Child::show() called

Compiler Error

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Leaderboard

1	A	alex	3972
2	M	Michael Allen	3903
3	N	Nathan	3861
4	S	Steve	3750
5	N	Nathaniel Waldschmidt	3000
5	A	alzlaz	3000
7	S	Syed	2805
8	S	Saiyed	2778
9	N	NP	2764
	M	Michael Kotwica	

Which of the following is true about inheritance in Java?

Which of the following is true about inheritance in Java?

```
1) Private methods are final.
2) Protected members are accessible within a package and inherited classes outside the
package.
3) Protected methods are final.
4) We cannot override private methods.
```

2, 3 and 4

1, 2 and 4

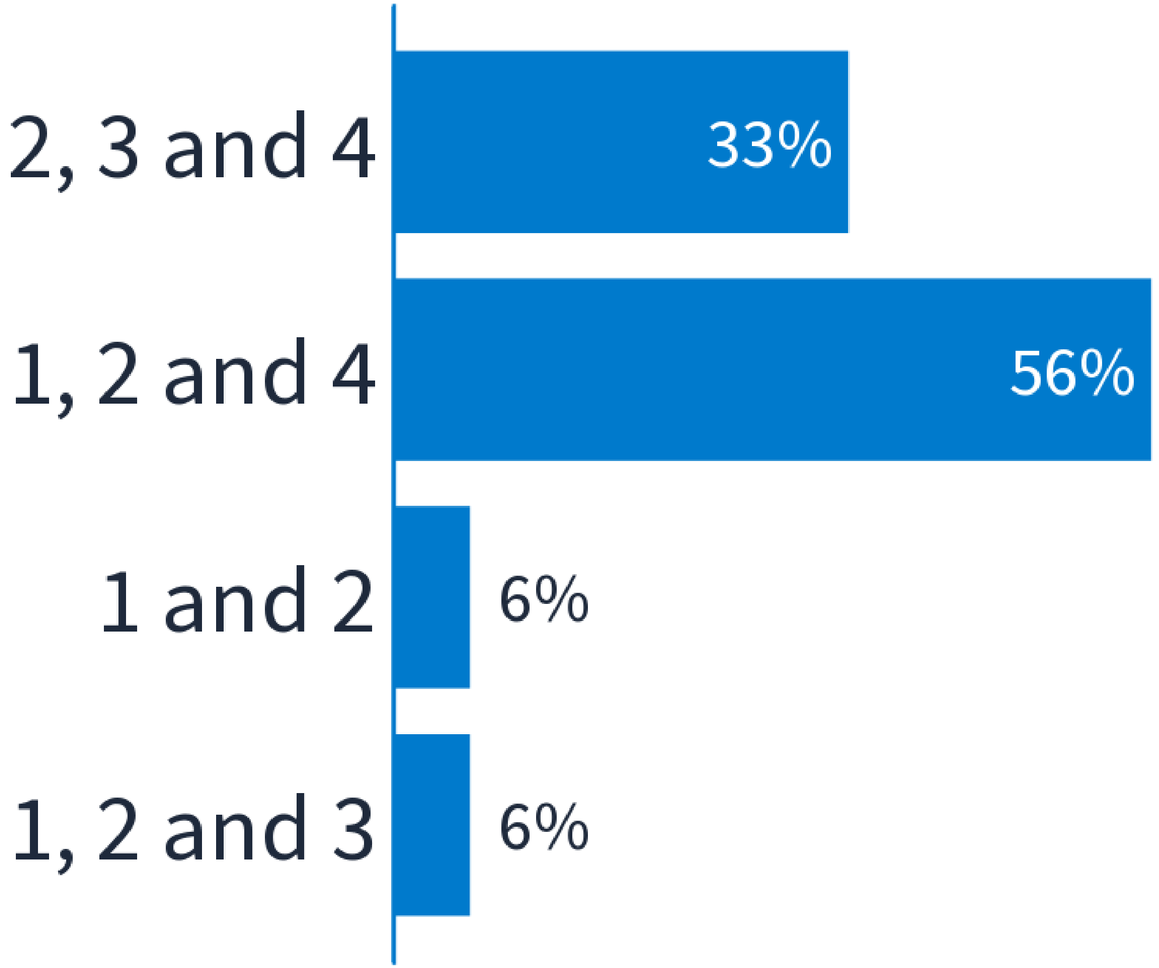
1 and 2

1, 2 and 3

Total Results: 0

Which of the following is true about inheritance in Java?

```
1) Private methods are final.
2) Protected members are accessible within a package and inherited classes outside the
package.
3) Protected methods are final.
4) We cannot override private methods.
```



Leaderboard

1	A	alex	3972
2	M	Michael Allen	3903
3	N	Nathan	3861
4	S	Steve	3750
5	N	Nathaniel Waldschmidt	3000
5	A	alzlaz	3000
7	S	Syed	2805
8	S	Saiyed	2778
9	N	NP	2764
	M	Michael Kotwica	

Output of following Java program?

Output of following Java program?

```java
class Parent {
    public void Print() {
        System.out.println("Parent");
    }
}

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

class Main{
    public static void DoPrint( Parent p ) {
        p.Print();
    }
    public static void main(String[] args) {
        Parent ob1 = new Parent();
        Parent ob2 = new Child();
        Child  ob3 = new Child();
        DoPrint(ob1);
        DoPrint(ob2);
        DoPrint(ob3);
    }
}
```

- Compiler Error
- Parent / Child / Child
- Parent / Parent / Child
- Parent / Child / Parent

Total Results: 0

Output of following Java program?

```java
class Parent {
    public void Print() {
        System.out.println("Parent");
    }
}

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

class Main{
    public static void DoPrint( Parent p ) {
        p.Print();
    }
    public static void main(String[] args) {
        Parent ob1 = new Parent();
        Parent ob2 = new Child();
        Child  ob3 = new Child();
        DoPrint(ob1);
        DoPrint(ob2);
        DoPrint(ob3);
    }
}
```

Compiler Error

Parent / Child / Child

Parent / Parent / Child

Parent / Child / Parent

Leaderboard

Any Question

??????????????????

How do you feel about the course?

Please Send Your Question or Feedback...

Top **New**