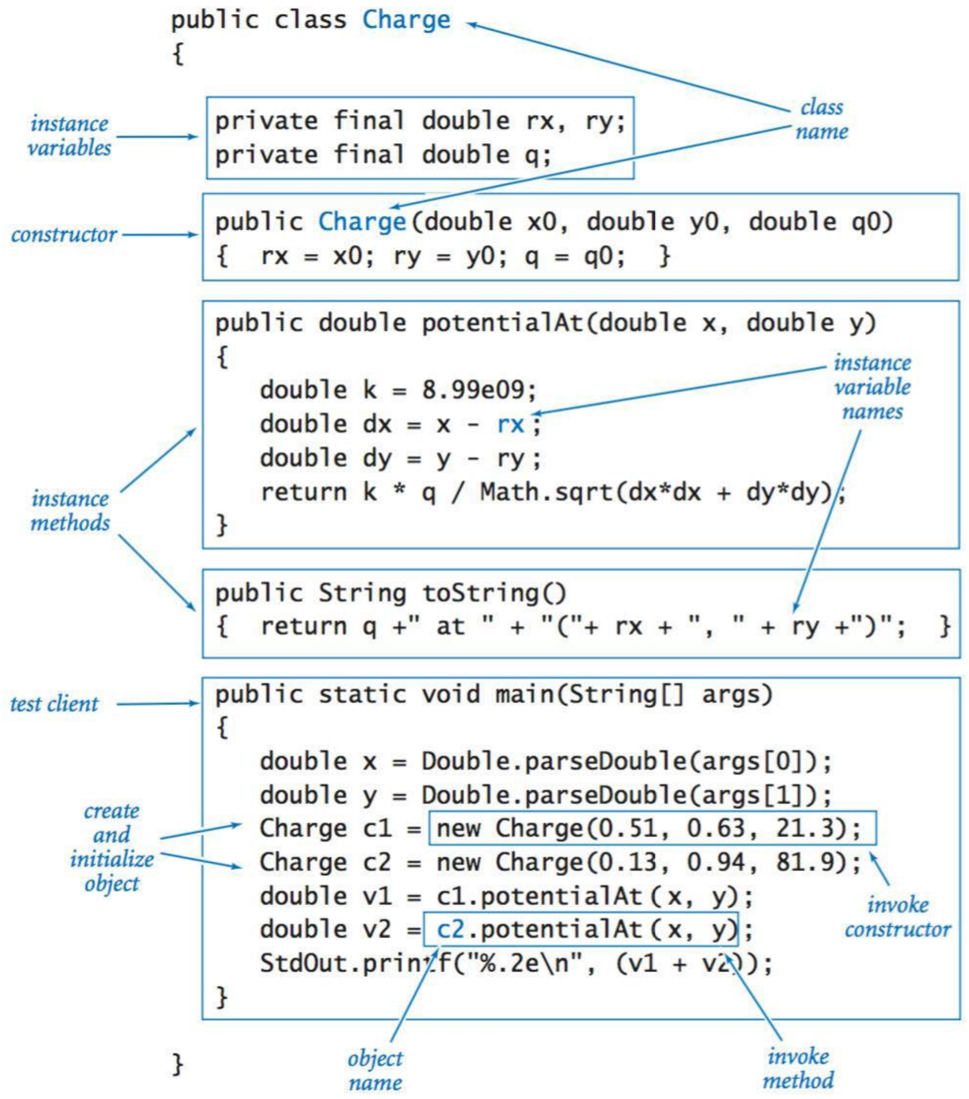
DEPAUL UNIVERSITY

SE 450 Object-Oriented Software Development

Object-oriented Programming : In-Depth

DEPAUL UNIVERSITY

Previous Lecture :OOP Building Blocks



```
public class Charge
{
   private final double rx, ry;
   private final double q;

   public Charge(double x0, double y0, double q0)
   {  rx = x0; ry = y0; q = q0;  }

   public double potentialAt(double x, double y)
   {
      double k = 8.99e09;
      double dx = x - rx;
      double dy = y - ry;
      return k * q / Math.sqrt(dx*dx + dy*dy);
   }

   public String toString()
   {  return q +" at " + "("+ rx + ", " + ry +")";  }

   public static void main(String[] args)
   {
      double x = Double.parseDouble(args[0]);
      double y = Double.parseDouble(args[1]);
      Charge c1 = new Charge(0.51, 0.63, 21.3);
      Charge c2 = new Charge(0.13, 0.94, 81.9);
      double v1 = c1.potentialAt(x, y);
      double v2 = c2.potentialAt(x, y);
      StdOut.printf("%.2e\n", (v1 + v2));
   }
}
```

Outline

- Object-oriented Programming : In-Depth
- OOP Concepts

Access modifiers

- Access modifiers
  - Public, private, protected, default
  - public > protected > package-private (or default) > private
- Public vs. Private
- Q&A
  - Why Main() method is always public?
  - What happens when no access modifier is defined?
- Example (Access Control) [AccessControlExample.java]
  - Accessing the public and private members

Getters and Setters

- How to access private members of a class ?
  - Public getter and setter methods
- Example [GetterSetterExample.java]
  - Creating getter and setter
  - Generate with IDE
  - Generate using lombox Project
    - https://projectlombok.org/

- Q&A
  - `What are the benefits of using Getter & Setter?

Initialization Block

- **Initialization Blocks**
  - Initialization block is called before constructor.
  - Non-static initialization blocks = instance initialization blocks (IIB)
  - Can be static or non-static

- **Examples**
  - InitializationBlockExample.java
  - StaticInitializationBlockExample.java

- **Q&A**
  - Why would you need initialization block when you have constructors?
  - Is it possible to have multiple initialization blocks?

Copying an Object

- Creating new instance from scratch
  - Costly, boring, time-consuming, and error-prone

- Copy constructor, Object cloning, Serialization, …

- Example
  - CopyObject.java
  - How to write your own copy constructor

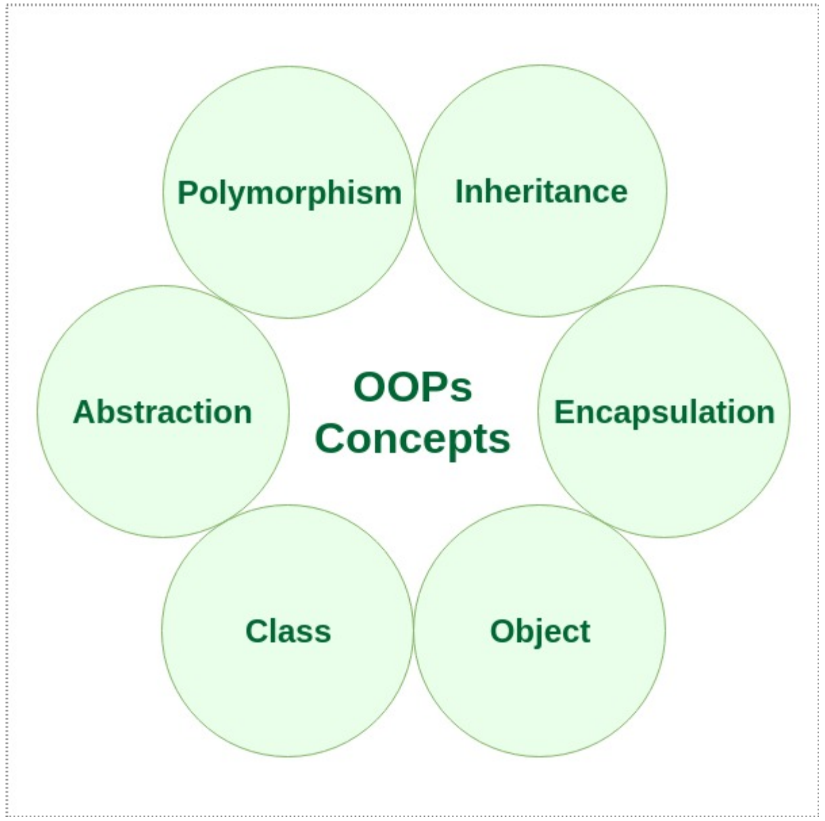
Outline

- Object-oriented Programming : In-Depth
- OOP Concepts

Concepts

- A software system may consist of many classes
  - Need to be managed

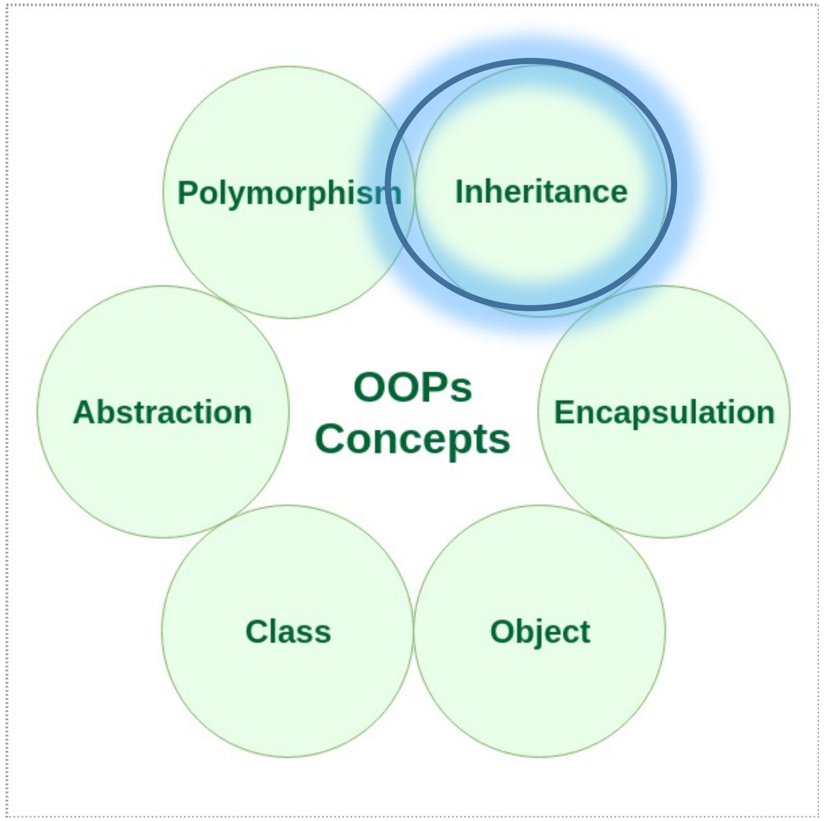
- Four main concepts
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism



DEPAUL UNIVERSITY

Concepts

- A software system may consist of many classes
  - Need to be managed
- Four main concepts
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism



Concept : Inheritance

- Allows to define a base class that provides specific functionality and to define derived classes.

- Keyword : extends

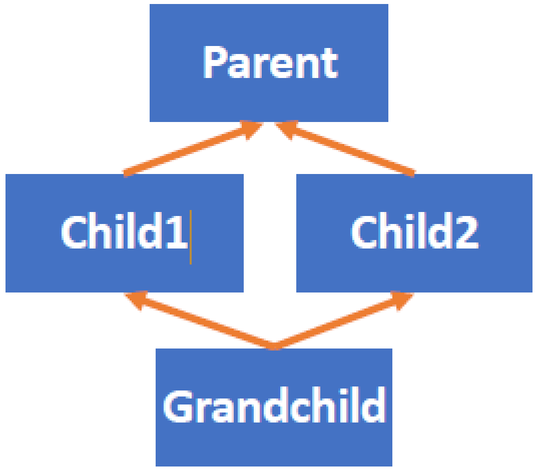
```
class derived-class extends base-class
{

//methods and fields

}
```

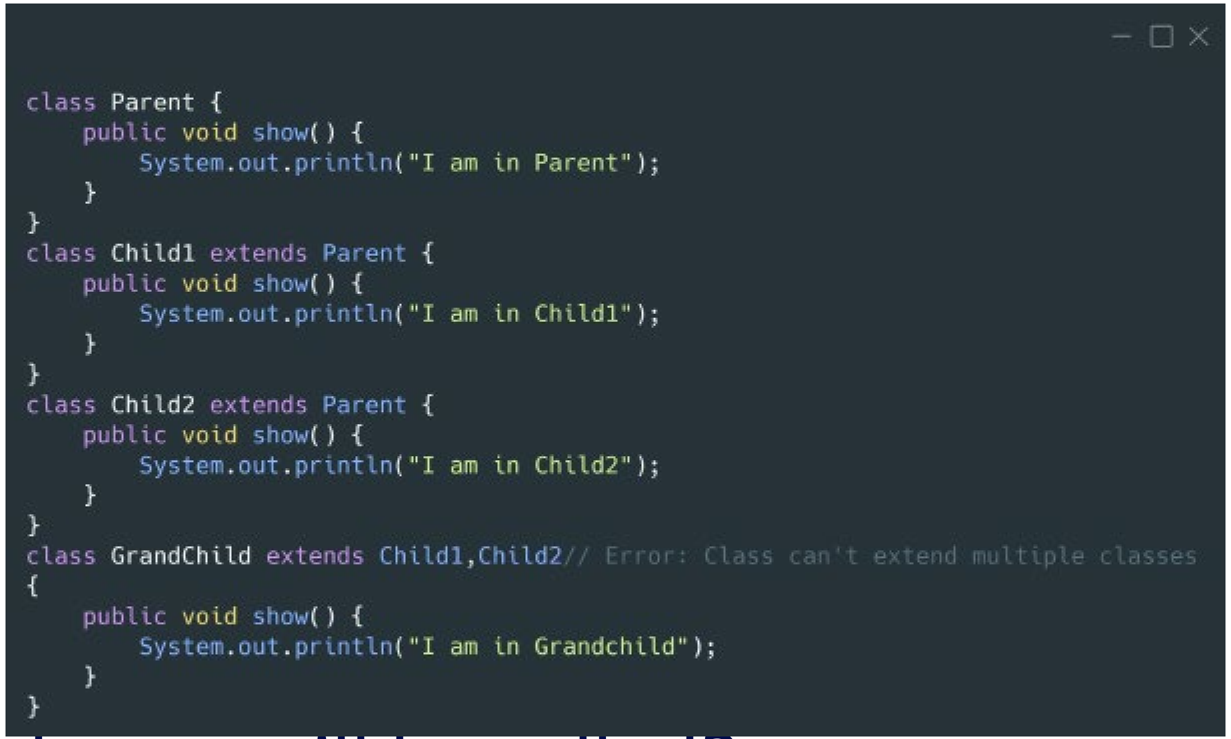
Types of Inheritance

- Single inheritance
- Hierarchical Inheritance
- Multi-level Inheritance
- Multiple Inheritance

Inheritance : Discussion

- Why Java does not support multiple inheritance via class?
  - Avoid ambiguity
  - Diamond problem



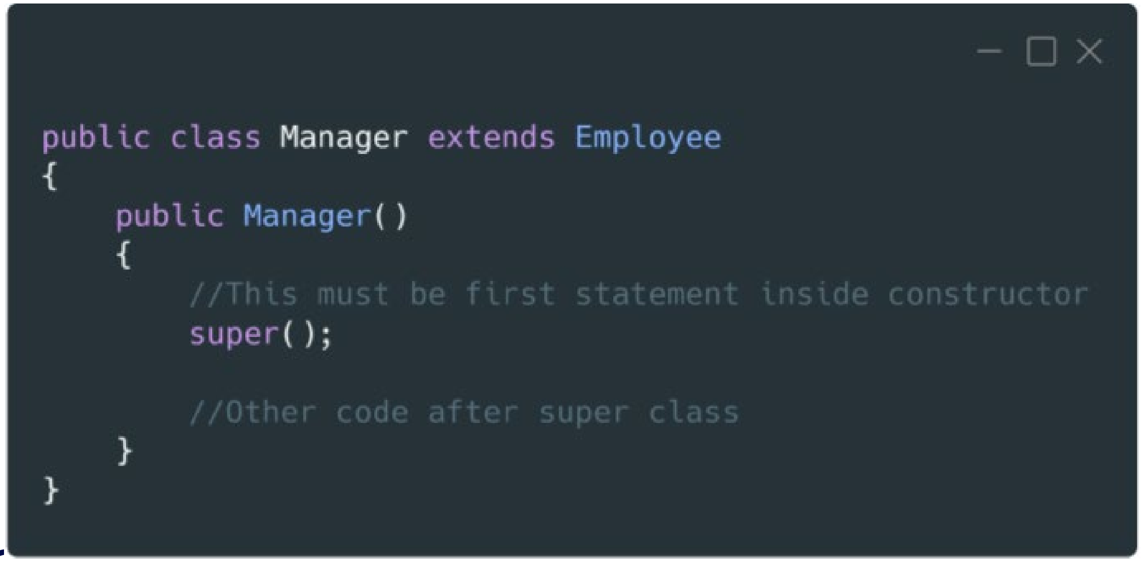


```
class Parent {
    public void show() {
        System.out.println("I am in Parent");
    }
}
class Child1 extends Parent {
    public void show() {
        System.out.println("I am in Child1");
    }
}
class Child2 extends Parent {
    public void show() {
        System.out.println("I am in Child2");
    }
}
class GrandChild extends Child1,Child2// Error: Class can't extend multiple classes
{
    public void show() {
        System.out.println("I am in Grandchild");
    }
}
```

- Which order constructors of the classes will be called?
  - See Example (Inheretance2.java)

Accessing inherited parent class members

- Super keyword
  - Reference variable used to refer immediate parent class object
    - immediate parent class instance variable
    - Invoke immediate parent class method
    - Invoke immediate parent class constructor:

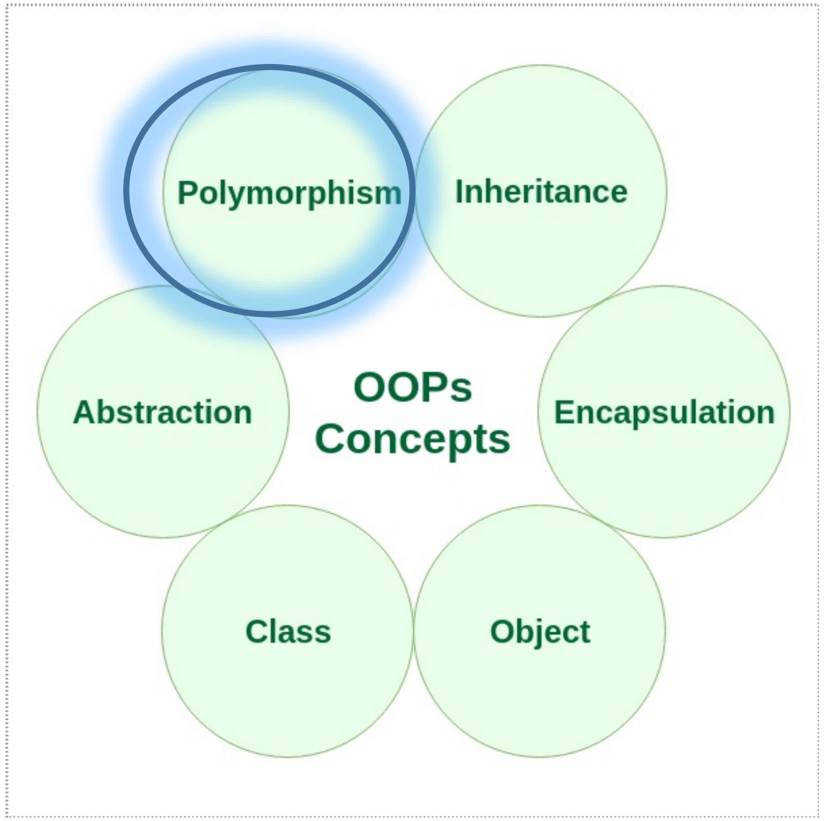


```
public class Manager extends Employee
{
    public Manager()
    {
        //This must be first statement inside constructor
        super();

        //Other code after super class
    }
}
```

- See Example : SuperVariable.java

Concepts

- A software system may consist of many classes
  - Need to be managed
- Four main concepts
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism



Concept : Polymorphism

- Polymorphism = “one name with many forms”
  - Compile time polymorphism
  - Runtime Polymorphism

- Compile time :
  - The compiler can bind the methods to the respective objects at compile time.
  - Method overloading : multiple functions with same name but different parameters
    - Method parameters can vary with a number, order, or the types of parameter
    - Example : Polymorphism1.java

Concept : Polymorphism

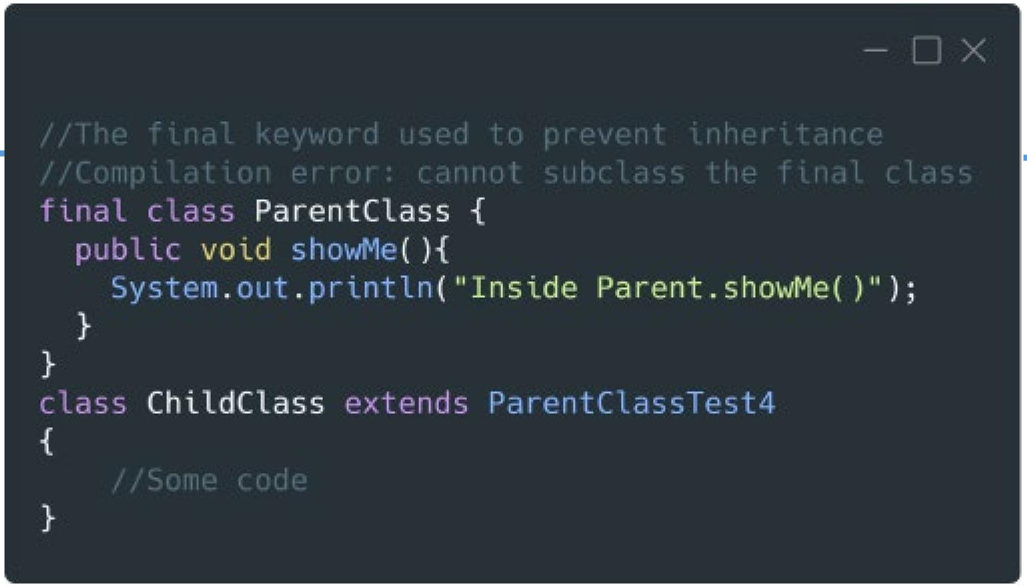
- Polymorphism = “one name with many forms”
  - Compile time polymorphism
  - Runtime Polymorphism

- Runtime polymorphism:
  - A function call to the overridden method is resolved at Runtime.
  - Method overriding : a derived class has a definition for one of the member functions of the base class
  - Example: Polymorphism2.java

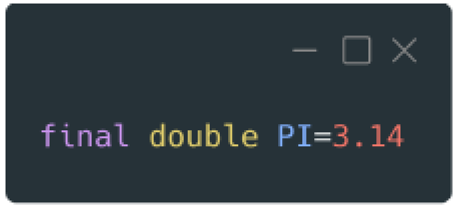
Final Keyword

- Prevent the inheritance process

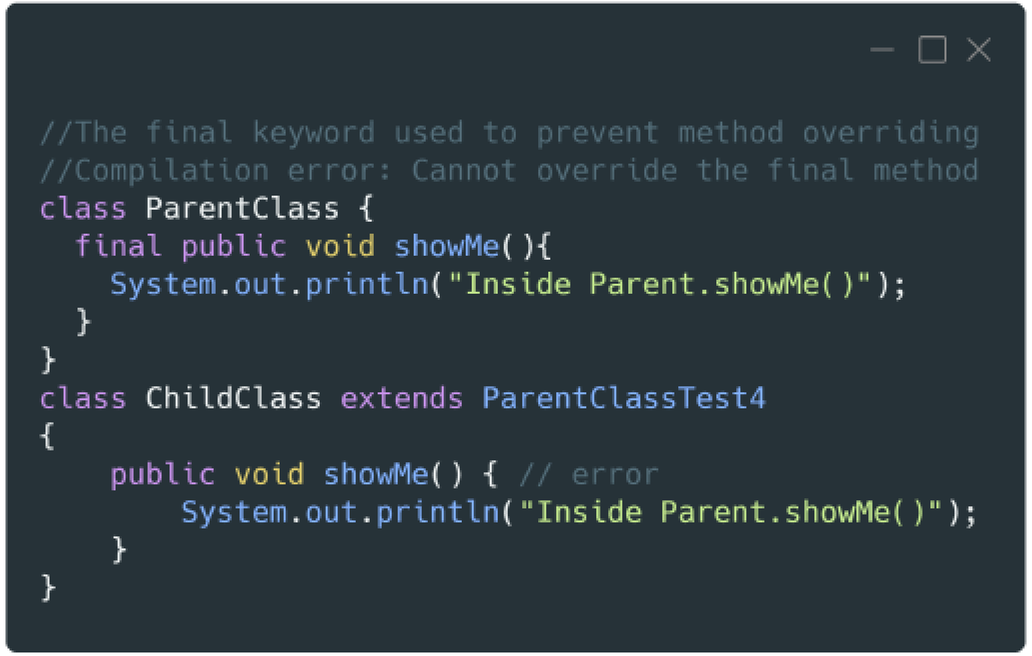
- Final Class : prevent inheritance
- Final Method: prevent method overriding
- Final Variable : Constant variable



```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
final double PI=3.14
```



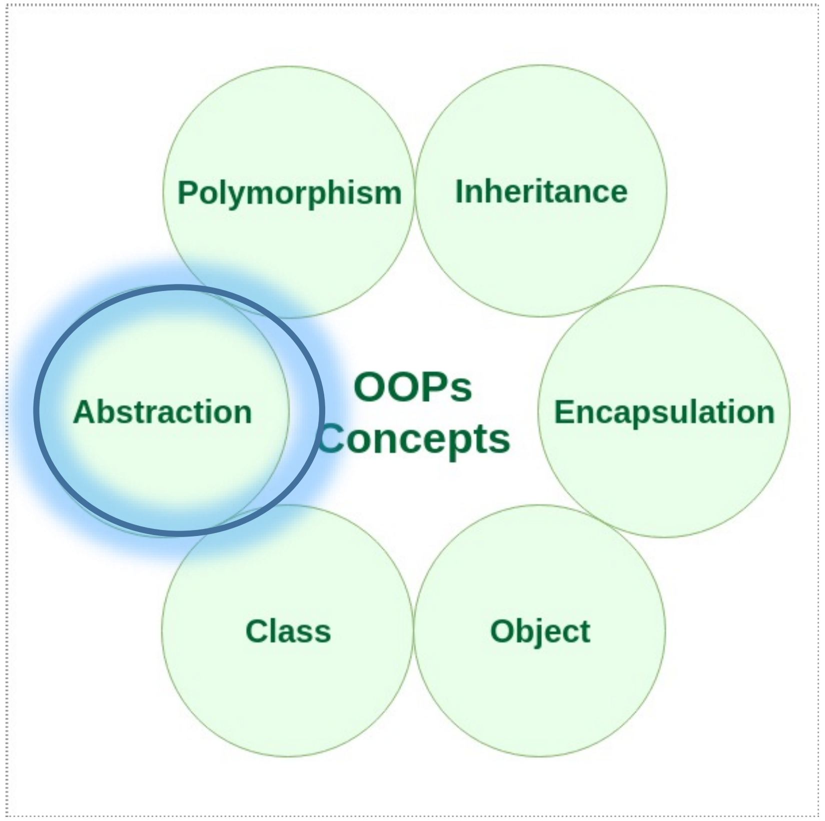
```java
//The final keyword used to prevent method overriding
//Compilation error: Cannot override the final method
class ParentClass {
  final public void showMe(){
    System.out.println("Inside Parent.showMe()");
  }
}
class ChildClass extends ParentClassTest4
{
    public void showMe() { // error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

- https://javaconceptoftheday.com/java-practice-questions-on-method-overloading-and-overriding/3/

Concepts

- A software system may consist of many classes
  - Need to be managed

- Four main concepts
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

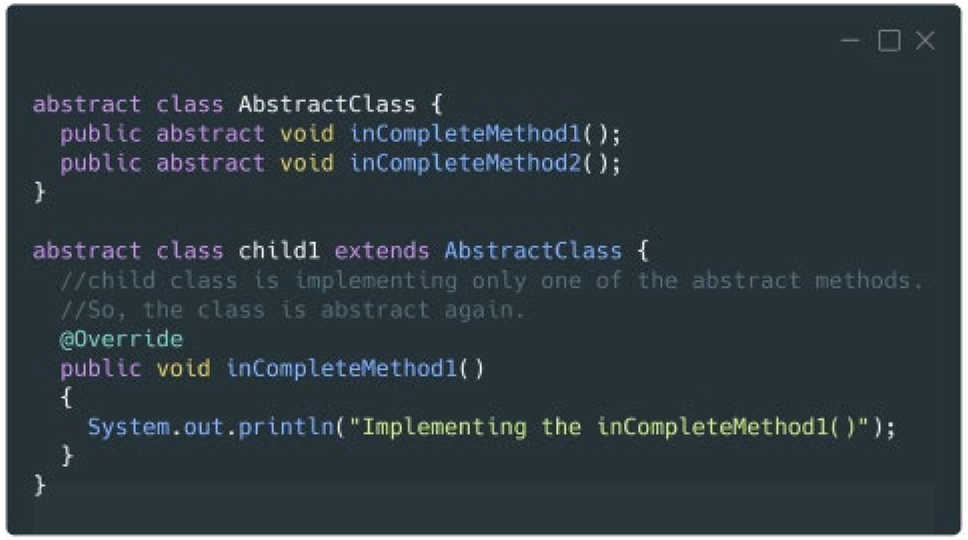


Abstraction

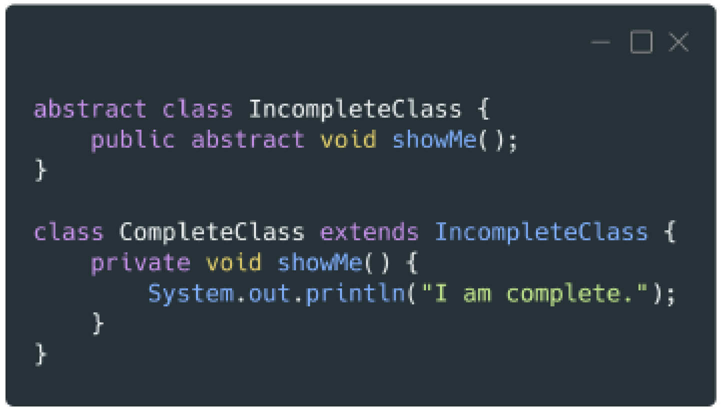
- Only the essential details are displayed to the user.
  - The trivial or the non-essentials units are not displayed to the user
- In java, abstraction is achieved by **interfaces** and **abstract classes**.
- An abstract class is a class that is declared with an abstract keyword.
- An abstract method = has declaration, no implementation.
- A method defined abstract must always be redefined in the subclass,
  - overriding OR make the subclass itself abstract.
- Any class that contains one or more abstract methods must also be declared with an abstract keyword.

Abstraction : Discussion

- Abstract class can contain fields.
- Must mark a class as abstract even with only one abstract method.
- You cannot create objects from an abstract class.
- If a class extends an abstract class, it must implement all the abstract methods.
  - Top Code Example
- A concrete class is a class that is not abstract
- You cannot reduce the visibility of an inherited method
  - Bottom Code Example
  - The access modifier of an overriding method must provide at least as much access as the overridden method itself.



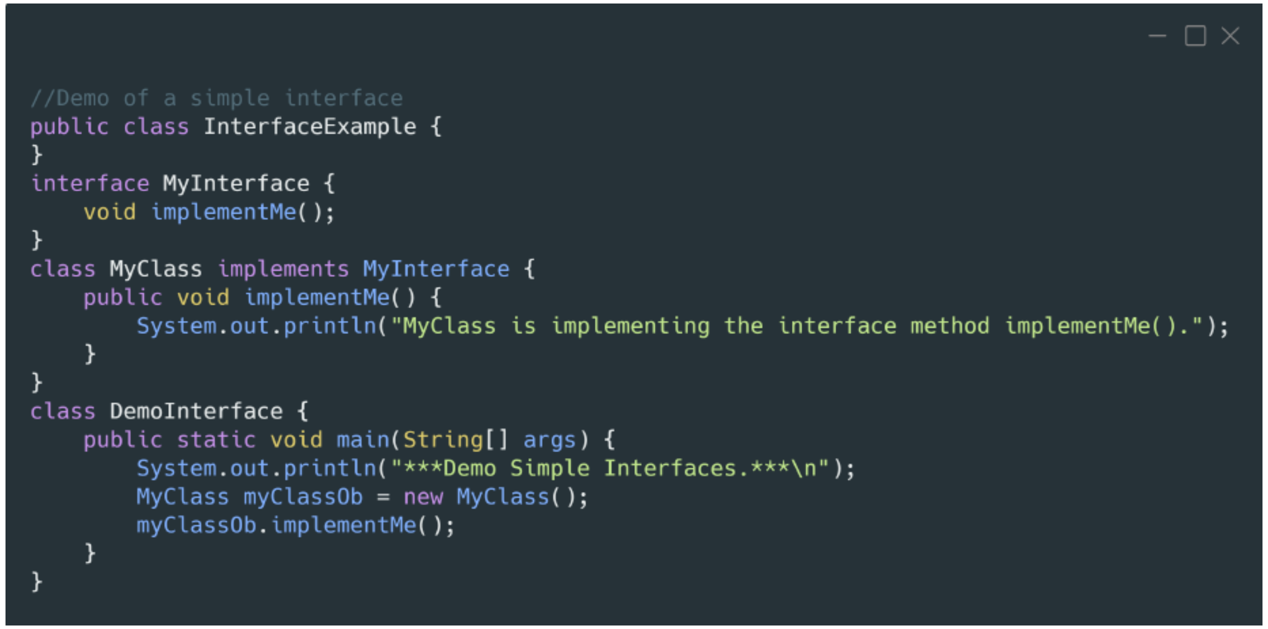
```java
abstract class AbstractClass {
  public abstract void inCompleteMethod1();
  public abstract void inCompleteMethod2();
}

abstract class child1 extends AbstractClass {
  //child class is implementing only one of the abstract methods.
  //So, the class is abstract again.
  @Override
  public void inCompleteMethod1()
  {
    System.out.println("Implementing the inCompleteMethod1()");
  }
}
```



```java
abstract class IncompleteClass {
    public abstract void showMe();
}

class CompleteClass extends IncompleteClass {
    private void showMe() {
        System.out.println("I am complete.");
    }
}
```

Interface

- A special type in Java
- Declares ***What*** to implement, not **How** to implement
- All methods are defined without body
- Syntax: interface MyInerface {}
- Using interface, we can support multiple inheritance in Java.



```
//Demo of a simple interface
public class InterfaceExample {
}
interface MyInterface {
    void implementMe();
}
class MyClass implements MyInterface {
    public void implementMe() {
        System.out.println("MyClass is implementing the interface method implementMe().");
    }
}
class DemoInterface {
    public static void main(String[] args) {
        System.out.println("***Demo Simple Interfaces.***\n");
        MyClass myClassOb = new MyClass();
        myClassOb.implementMe();
    }
}
```

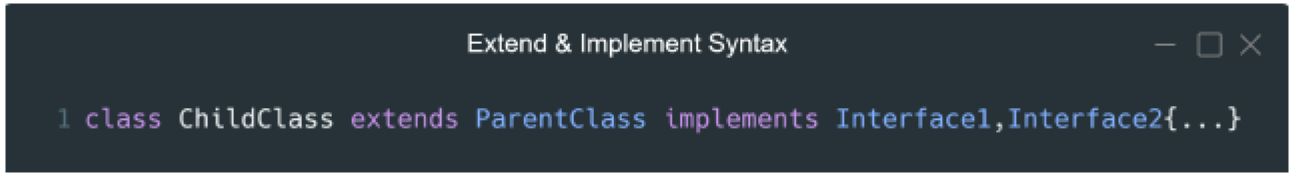
Interface : Discussion

- Abstract class using interface
  - The class that is using the interface must implement all the methods. If not, it is an abstract.
  - See example

- The class that is using the interface must implement all the methods. If not, it is an abstract.

- Extend and implement at the same time

- Positional notation:
  - Extend before Implement


Extend & Implement Syntax

```
class ChildClass extends ParentClass implements Interface1,Interface2{...}
```

- Following this design, the compiler knows about the parent class first and can point out any compilation errors in the parent class.

- Questions ?