Implementation Inheritance

An introduction to the Java Programming Language

Produced by:

Eamonn de Leastar (edeleastar@wit.ie)

Dr. Siobhan Drohan (sdrohan@wit.ie)

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Essential Java

- **Overview**
  - Introduction
  - Syntax
  - Basics
  - Arrays
- **Classes**
  - Classes Structure
  - Static Members
  - Commonly used Classes
- **Control Statements**
  - Control Statement Types
  - If, else, switch
  - For, while, do-while

- **Inheritance**
  - Class hierarchies
  - Method lookup in Java
  - Use of this and super
  - Constructors and inheritance
  - Abstract classes and methods
  - Interfaces
- **Collections**
  - ArrayList
  - HashMap
  - Iterator
  - Vector
  - Enumeration
  - Hashtable

- **Exceptions**
  - Exception types
  - Exception Hierarchy
  - Catching exceptions
  - Throwing exceptions
  - Defining exceptions
  - Common exceptions and errors
- **Streams**
  - Stream types
  - Character streams
  - Byte streams
  - Filter streams
  - Object Serialization

Agenda

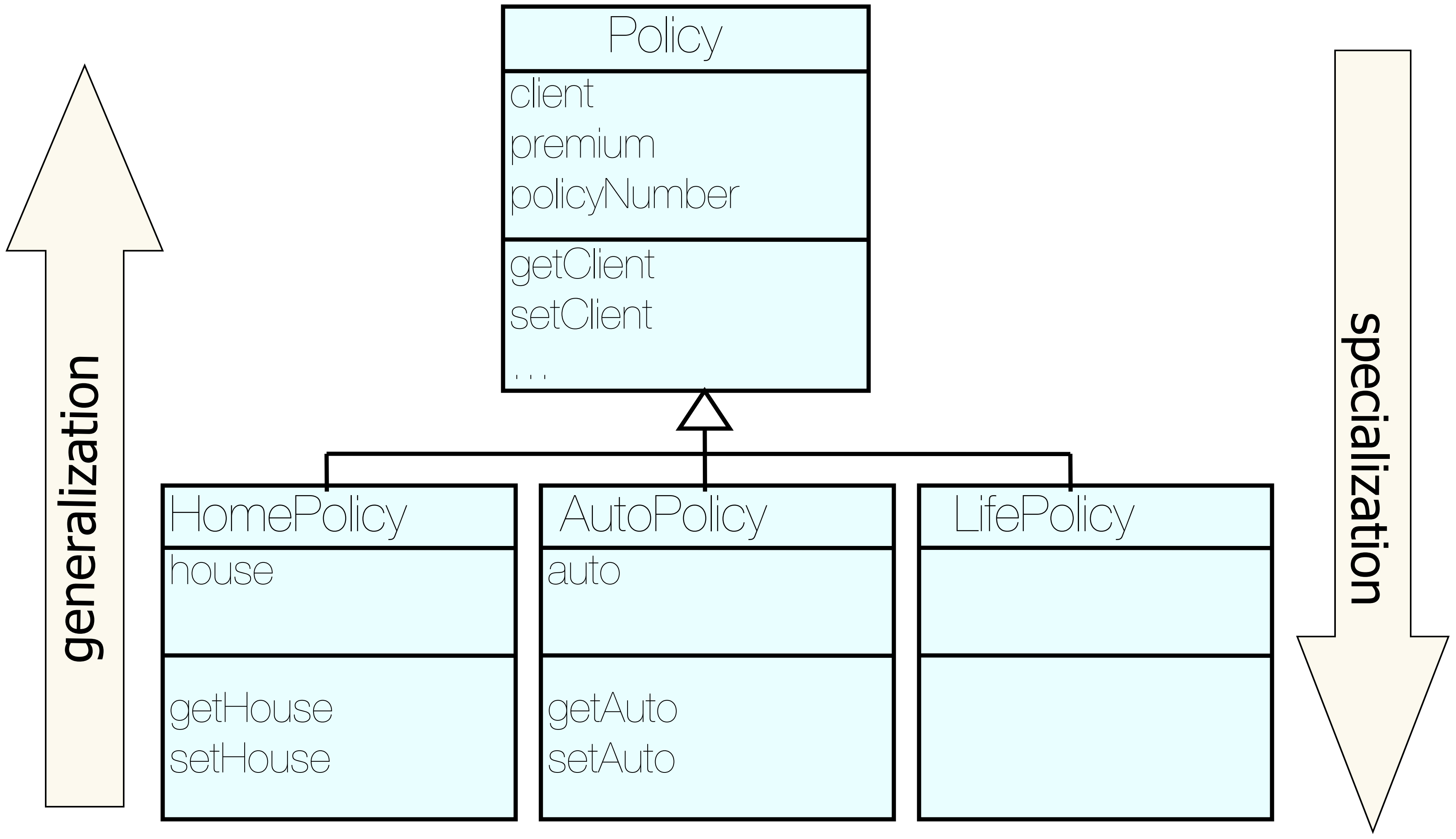
- What is inheritance?
- Implementation Inheritance
  - Method lookup in Java
  - Use of this and super
  - Constructors and inheritance
  - Abstract classes and methods

What is Inheritance?

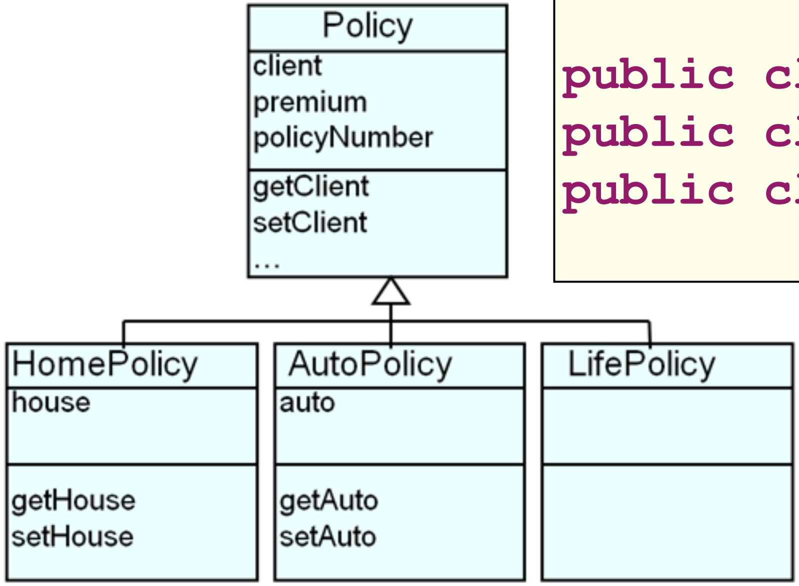
- Inheritance is one of the primary object-oriented principles.

Implementation Inheritance	Interface Inheritance
• Promotes reuse. • Commonalities are stored in a parent class (superclass). • Commonalities are shared between children classes (subclasses).	• Mechanism for introducing ***Types*** into java design. • Classes can support more than one interface, i.e. be of more than one ***type.***

Implementation Inheritance

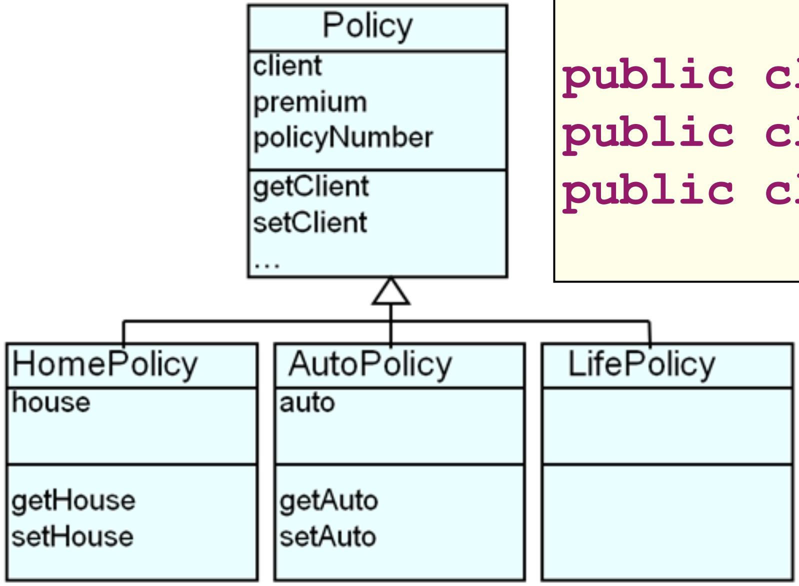


Defining Inheritance



```
public class Policy {…

public class HomePolicy extends Policy{…
public class AutoPolicy extends Policy{…
public class LifePolicy extends Policy{…
```

Defining Inheritance



```
public class Policy {…

public class HomePolicy extends Policy{…
public class AutoPolicy extends Policy{…
public class LifePolicy extends Policy{…
```

- If the class does not explicitly specify a superclass, its superclass is Object class.

```
public class Policy{…
```

=

```
public class Policy extends Object{…
```

Variables and Inheritance

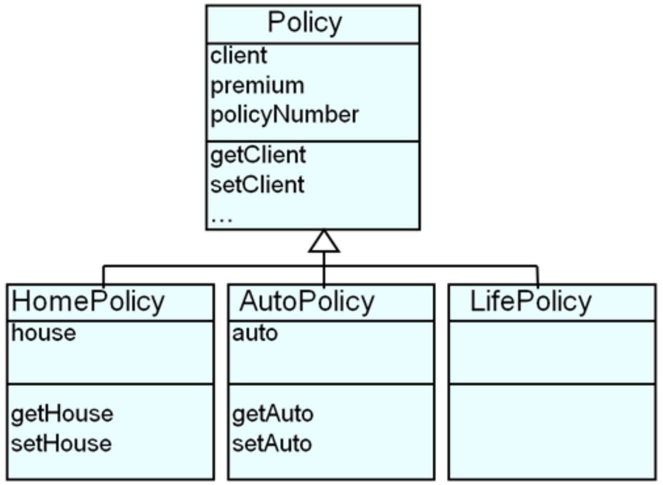
- Variables can be declared against the superclass, and assigned objects of the subclass.

```
Policy policy;
policy = new Policy();
```

```
Policy policy;
policy = new HomePolicy();
```

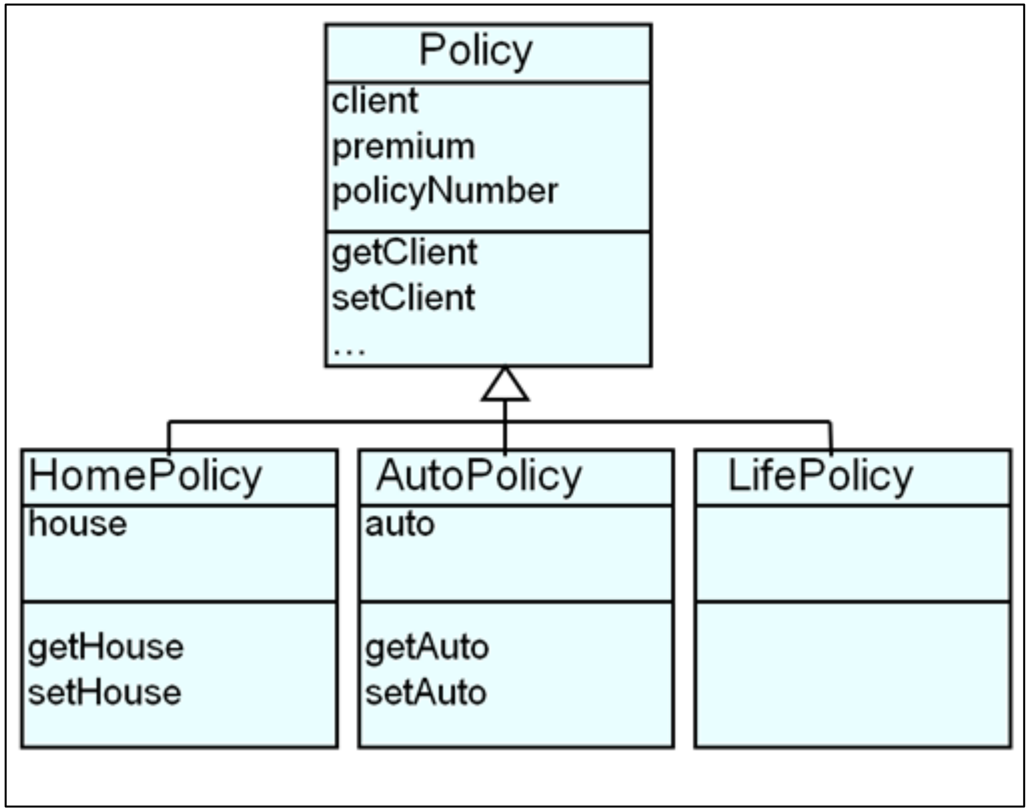
```
Policy policy;
policy = new AutoPolicy();
```

```
Policy policy;
policy = new LifePolicy();
```



What is Inherited?

- Subclasses inherit from superclass:
  - Fields (instance variables) i.e. data.
  - Methods i.e. behaviours.

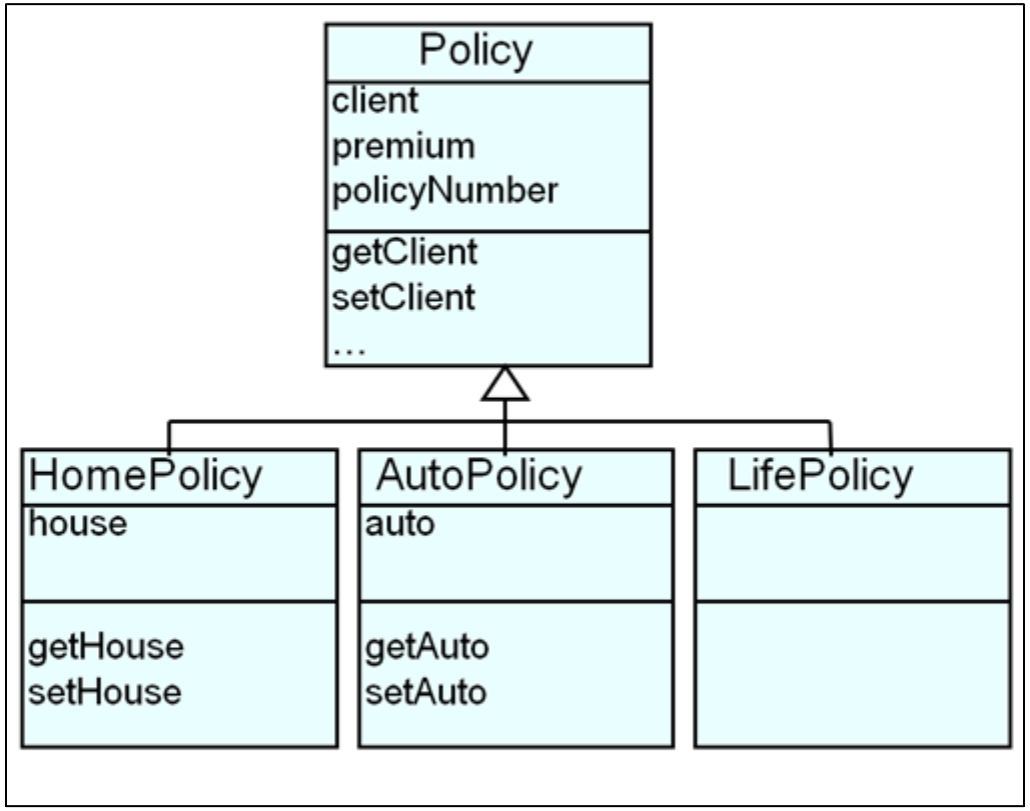


Inheriting Fields

- All fields from superclasses are inherited by a subclass.
- Inheritance goes all the way up the hierarchy.

Policy:
client
premium
policyNumber

HomePolicy:
client
premium
policyNumber
house

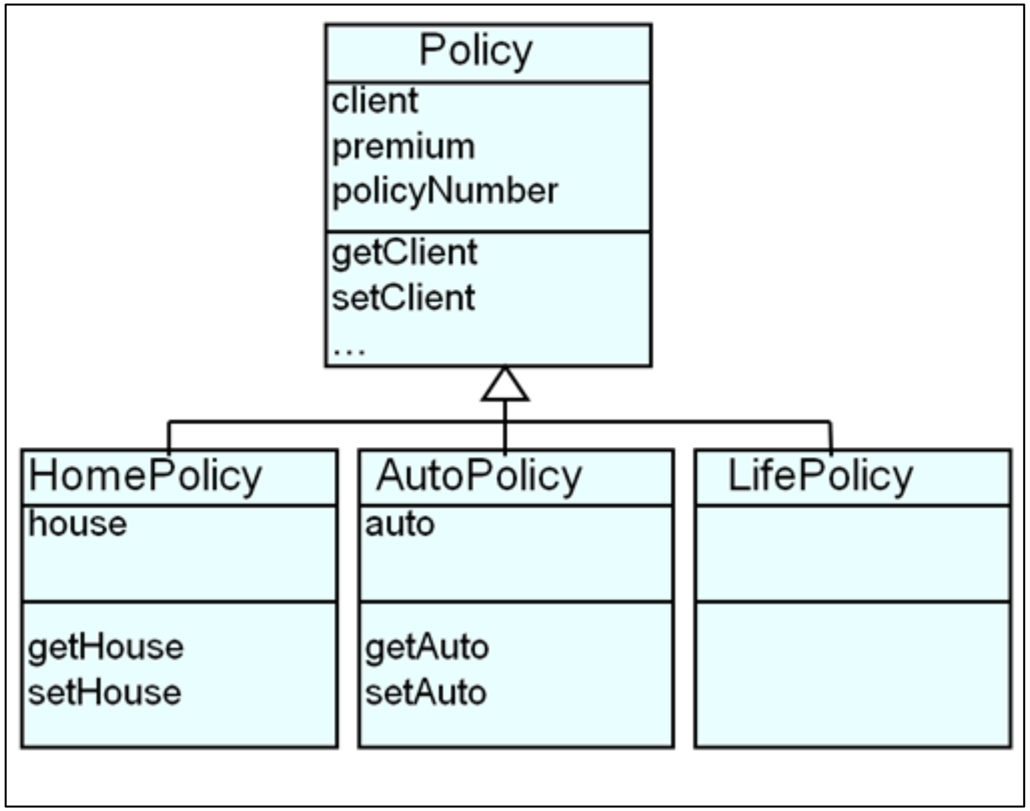


Inheriting Methods

- All methods from superclasses are inherited by a subclass
- Inheritance goes all the way up the hierarchy

Policy:
getClient
setClient
. . .

HomePolicy:
getClient
setClient
. . .
getHouse
setHouse

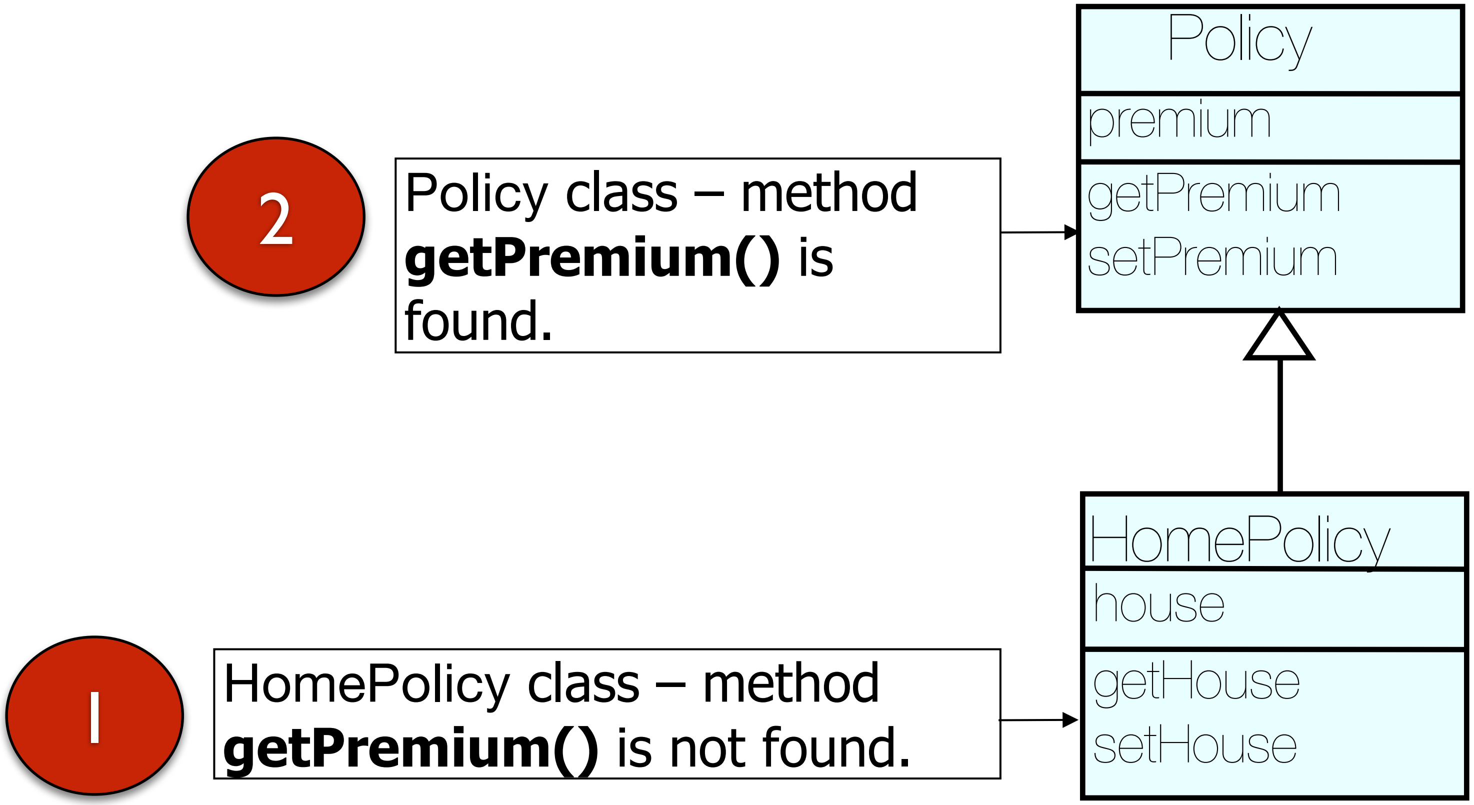


Agenda

- What is inheritance?
- Implementation Inheritance
    - Method lookup in Java
    - Use of this and super
    - Constructors and inheritance
    - Abstract classes and methods

Method Lookup

```
HomePolicy homePolicy = new HomePolicy();

homePolicy.getPremium();
```


2
Policy class – method **getPremium()** is found.
Policy
premium
getPremium
setPremium
HomePolicy
house
getHouse
setHouse
1
HomePolicy class – method **getPremium()** is not found.

this **vs.** super

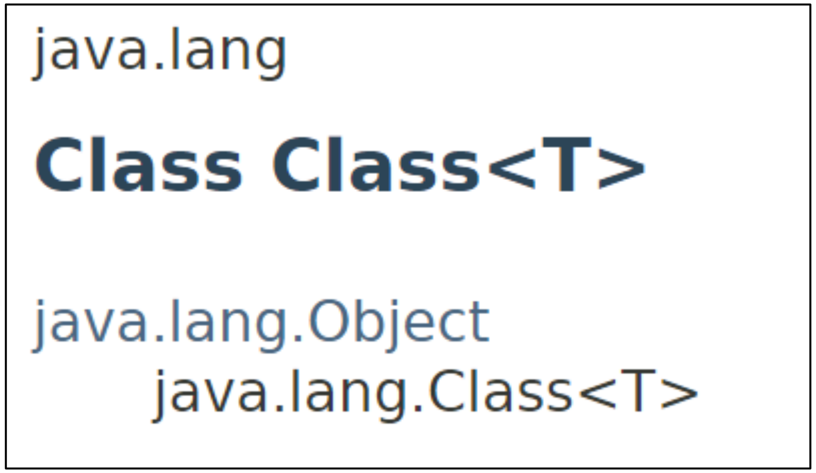
- They are both names of the receiver object:
  - **this**: used for pointing to the current class instance.
  - **super**: lookup begins in the superclass of the class where super was defined.

```
class HomePolicy extends Policy
{
  private int instalments;
  private String house;

  public void setInstalments (int instalments){
    this.instalments = instalments;
  }

  public void print(){
    super.print();
    System.out.println("for house " + getHouse().toString();
  }

  //…
}
```

getClass()

- getClass()
  - Method in java.lang.Object.
  - It returns the runtime class of the receiver object e.g.

    com.example.HomePolicy



java.lang

Class Class<T>

java.lang.Object
    java.lang.Class<T>

- getClass().getName()
  - Method in java.lang.Class.
  - It returns the name of the class or interface of the receiver object e.g.

    HomePolicy

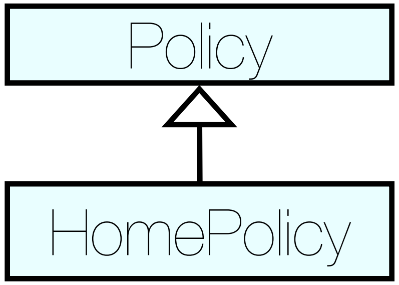
```
class Policy
{
  //…
  public void print()
  {
    System.out.println("A " + getClass().getName() + ", $" + getPremium());
  }
  //..
}
```

```
Policy p = new Policy();
p.print();
```

```
A Policy, $1,200.00
```

16

```
class HomePolicy extends Policy
{
  //…
  public void print()
  {
    super.print();
    System.out.println("for house " + getHouse().toString();
  }
  //…
}
```

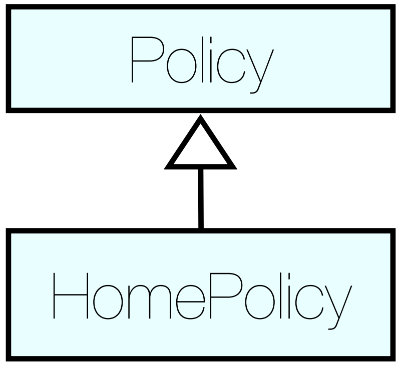


```
HomePolicy h = new HomePolicy();
h.print();
```

```
A HomePolicy, $1,200.00
for house 200 Great Street
```

Method Overriding

- If a class defines the same method as its superclass, it is said that the method is overridden
- Method signatures must match



```
//Method in the Policy class
public void print()
{
  System.out.println("A " + getClass().getName() + ", $" +  getPremium());
}
```

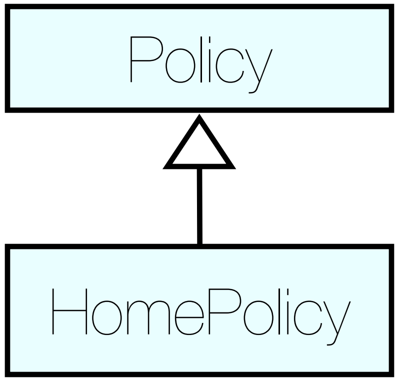
```
//Overridden method in the HomePolicy class
public void print()
{
 super.print();
 System.out.println("for house " + getHouse().toString();
}
```

Agenda

- What is inheritance?
- Implementation Inheritance
    - Method lookup in Java
    - Use of this and super
    - Constructors and inheritance
    - Abstract classes and methods

Constructors and Inheritance

```
public Policy(double premium, Client aClient, String policyNumber)
{
    this.premium      = premium;
    this.policyNumber = policyNumber;
    this.client       = aClient;
}
```



```
public HomePolicy(double premium,
                  Client aClient,
                  String policyNumber,
                  House  aHouse)
{
    super(premium, aClient, policyNumber);
    this.house = aHouse;
}
```

- First line must be a call to the super constructor

Constructors and Inheritance

- Constructors are not inherited by the subclasses.
- If the call is not coded explicitly then an implicit zero-argument super() is called.
- If the superclass does not have a zero-argument constructor, this causes an error.
- Adopting this approach eventually leads to the Object class constructor that creates the object.

Overview: Road Map

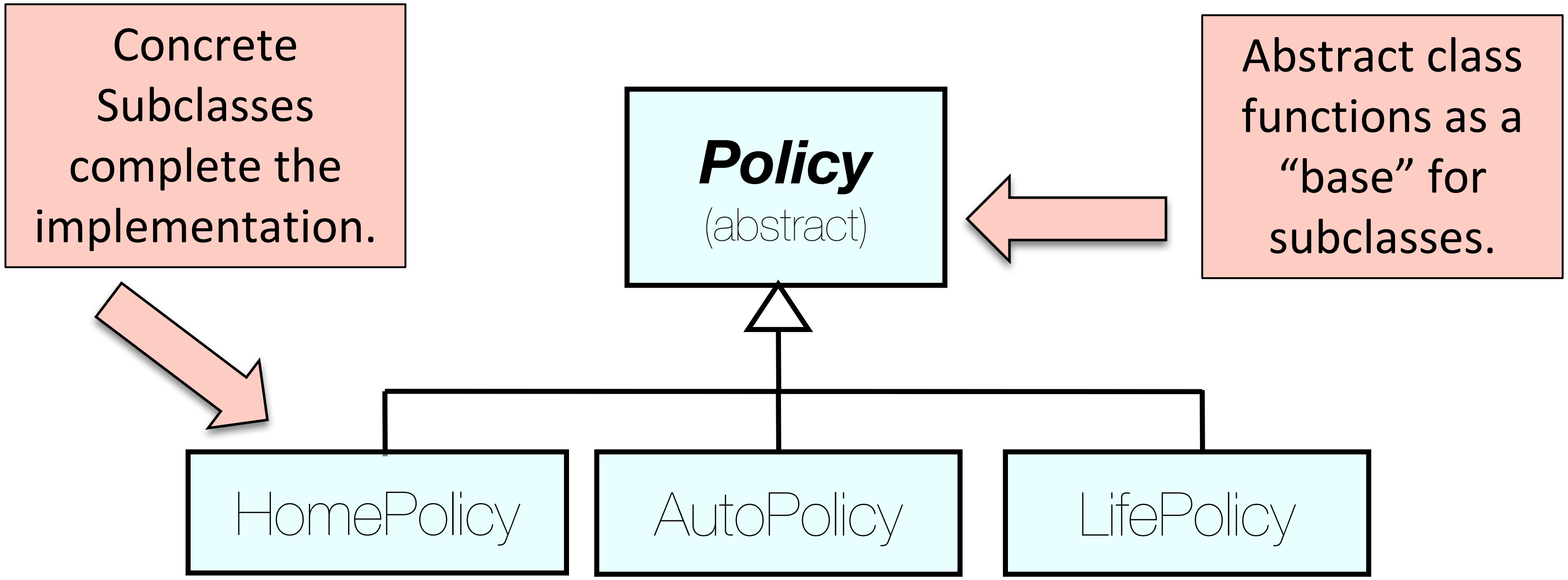
- What is inheritance?
- Implementation Inheritance
  - Method lookup in Java
  - Use of this and super
  - Constructors and inheritance
  - Abstract classes and methods

Defining Abstract Classes

```
public abstract class Policy {

    // can contain zero or more abstract methods.
    // a class that has an abstract method must be declared abstract.
    // cannot create an instance of this abstract class.
}
```

Defining Abstract Classes

```
public abstract class Policy {

    // can contain zero or more abstract methods.
    // a class that has an abstract method must be declared abstract.
    // cannot create an instance of this abstract class.
}
```

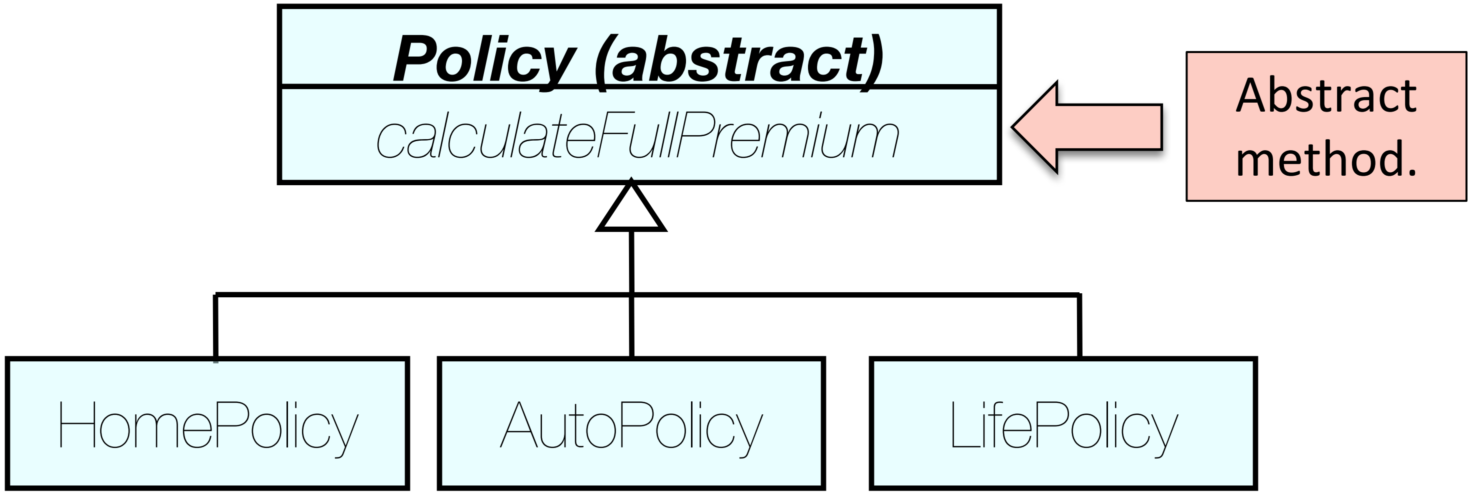

Concrete Subclasses complete the implementation.
Policy
(abstract)
Abstract class functions as a "base" for subclasses.
HomePolicy
AutoPolicy
LifePolicy

Defining Abstract Methods

```
public abstract class Policy
{
   // abstract classes can contain concrete methods as well.
   // abstract classes are not required to have abstract methods.

   /* each subclass must have a concrete implementation of the abstract
     method, or make themselves abstract. */

  public abstract void calculateFullPremium();
}
```

Defining Abstract Methods

```
public abstract class Policy
{
   // abstract classes can contain concrete methods as well.
   // abstract classes are not required to have abstract methods.

   /* each subclass must have a concrete implementation of the abstract
     method, or make themselves abstract. */

  public abstract void calculateFullPremium();
}
```



Defining Abstract Methods

```java
public class HomePolicy extends Policy
{
   //…
   public void calculateFullPremium()
   {
      //calculation may depend on a criteria about the house
   }
}
```

```java
public class AutoPolicy extends Policy
{
   //…
   public void calculateFullPremium()
   {
      //calculation may depend on a criteria about the auto
   }
}
```

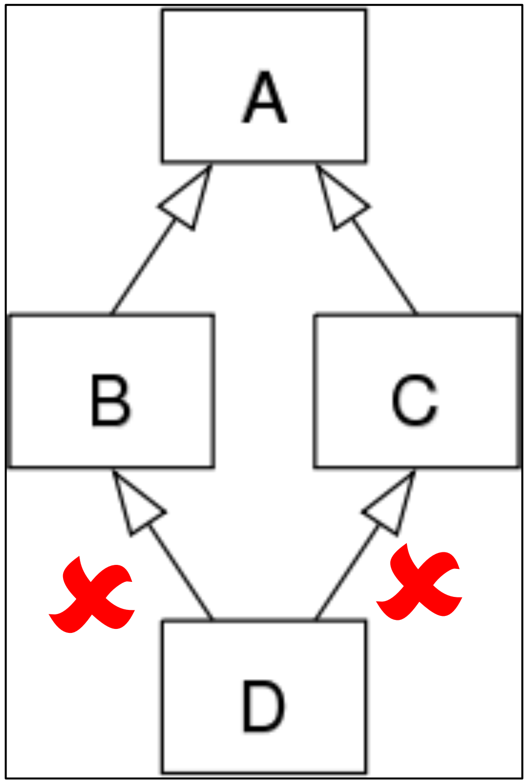
```java
public class LifePolicy extends Policy
{
   //…
   public void calculateFullPremium()
   {
      //calculation may depend on a criteria about the client
   }
}
```

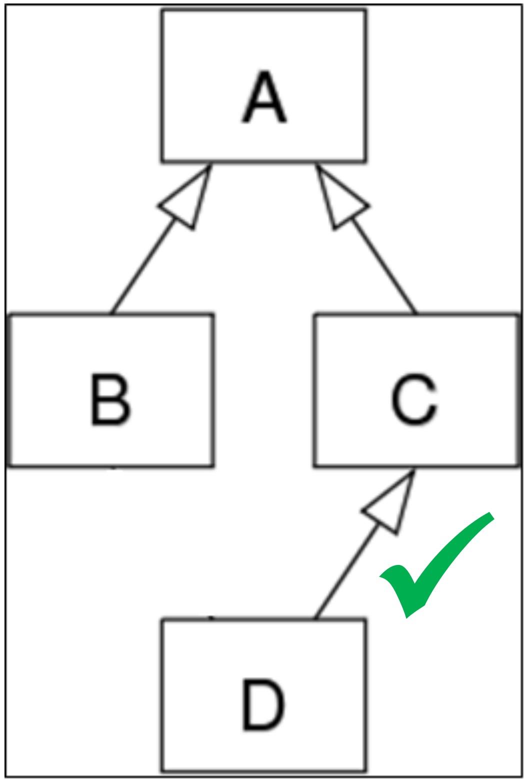
All subclasses must implement all abstract methods

Summary

- What is inheritance?
- Implementation Inheritance
    - Method lookup in Java
    - Use of this and super
    - Constructors and inheritance
    - Abstract classes and methods

Multiple Inheritance ?





- Not supported in Java. WHY?

Thought Experiment: Multiple Inheritance

- Let’s pretend that Java allows multiple inheritance

http://javacodeonline.blogspot.ie/2009/08/deadly-diamond-of-death.html

Thought Experiment: Multiple Inheritance

```
public abstract class AbstractSuperClass{
      abstract void do();
}
```

http://javacodeonline.blogspot.ie/2009/08/deadly-diamond-of-death.html

Thought Experiment: Multiple Inheritance

```
public abstract class AbstractSuperClass{
        abstract void do();
}
```

```
public class ConcreteOne extends AbstractSuperClass{
    void do(){
        System.out.println("I am testing multiple Inheritance");
    }
}
```

Thought Experiment: Multiple Inheritance

```
public abstract class AbstractSuperClass{
        abstract void do();
}
```

```
public class ConcreteOne extends AbstractSuperClass{
    void do(){
         System.out.println("I am testing multiple Inheritance");
    }
}
```

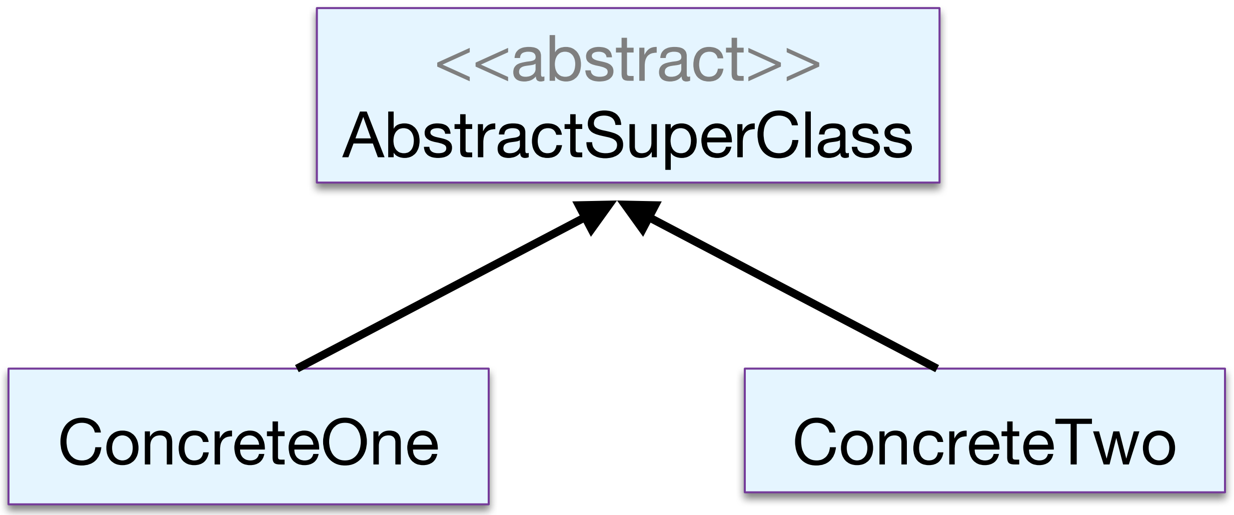
```
public class ConcreteTwo extends AbstractSuperClass{
      void do(){
          System.out.println("I will cause the Deadly Diamond of Death");
     }
}
```

Each class provides their own implementation of ***void do()***

http://javacodeonline.blogspot.ie/2009/08/deadly-diamond-of-death.html

Thought Experiment: Multiple Inheritance

- So far, our class diagram looks like this:



- No problems, yet…

http://javacodeonline.blogspot.ie/2009/08/deadly-diamond-of-death.html

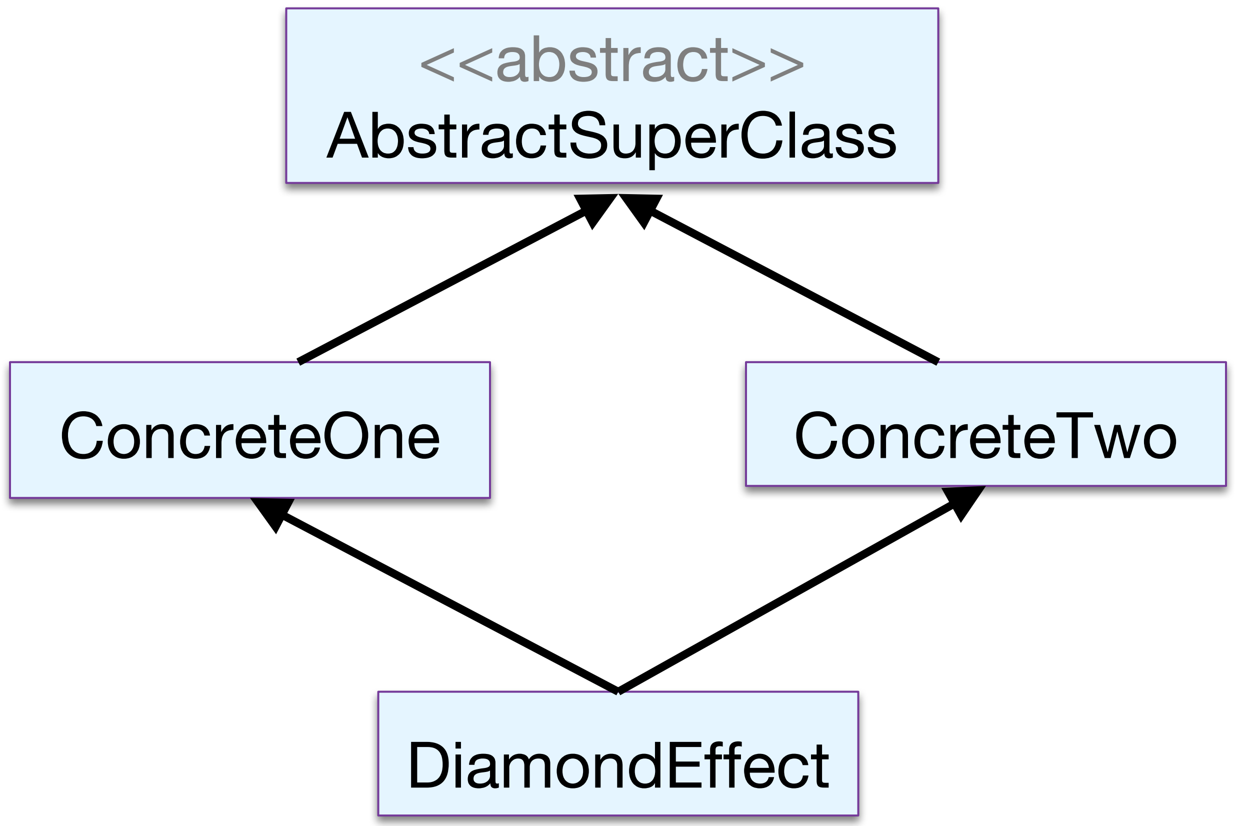
Thought Experiment: Multiple Inheritance

- Now, if multiple inheritance were allowed, a fourth class comes into picture which **extends** the above two concrete classes.

```
public class DiamondEffect extends ConcreteOne, ConcreteTwo{
     //Some methods of this class
}
```

Thought Experiment: Multiple Inheritance

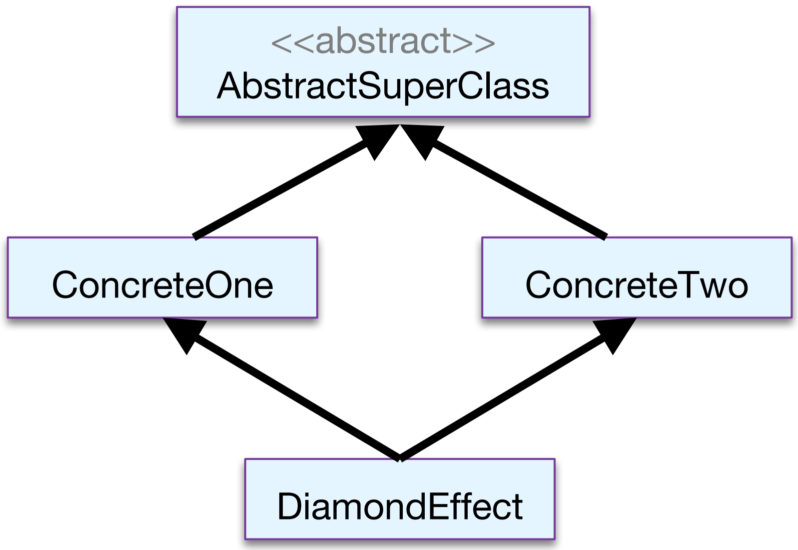
- Diamond shape class diagram



http://javacodeonline.blogspot.ie/2009/08/deadly-diamond-of-death.html

Thought Experiment: Multiple Inheritance

- The DiamondEffect class inherits all the methods of the parent classes.
- BUT we have a common method, *void do()*, in the two concrete classes, each with a different implementation.
- So which void do() implementation will be used for the DiamondEffect class as it inherits both these classes?



http://javacodeonline.blogspot.ie/2009/08/deadly-diamond-of-death.html

Deadly Diamond of Death

Actually this is a critical issue that the java designers wanted to avoid, so, the result was…

Multiple Inheritance

(although it is supported in C++ via Virtual Base class feature)

http://javacodeonline.blogspot.ie/2009/08/deadly-diamond-of-death.html