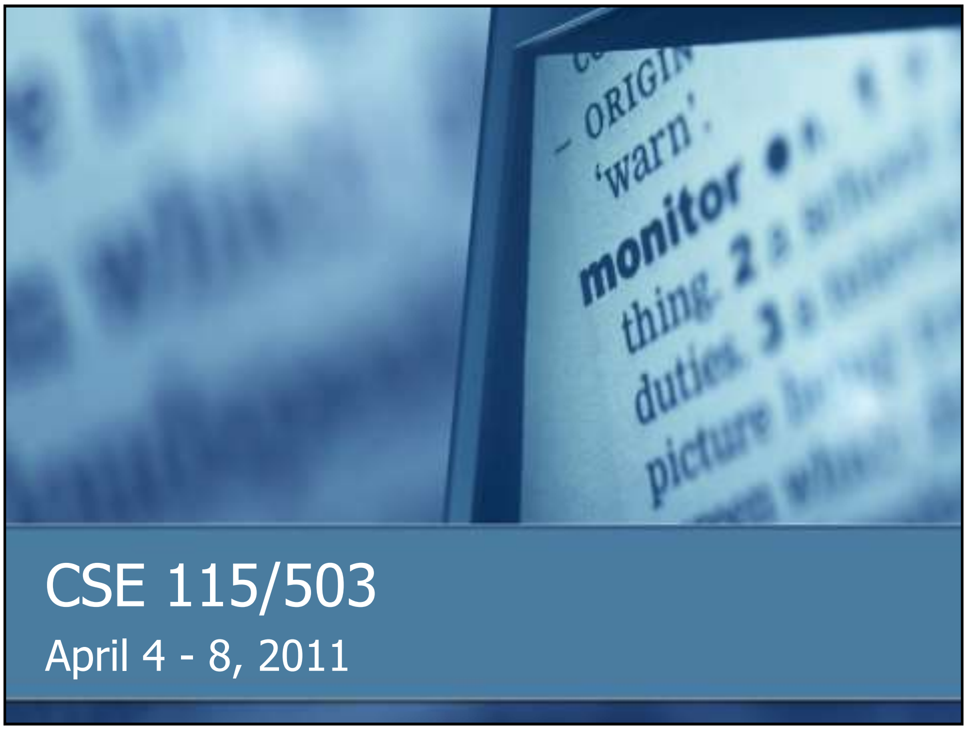
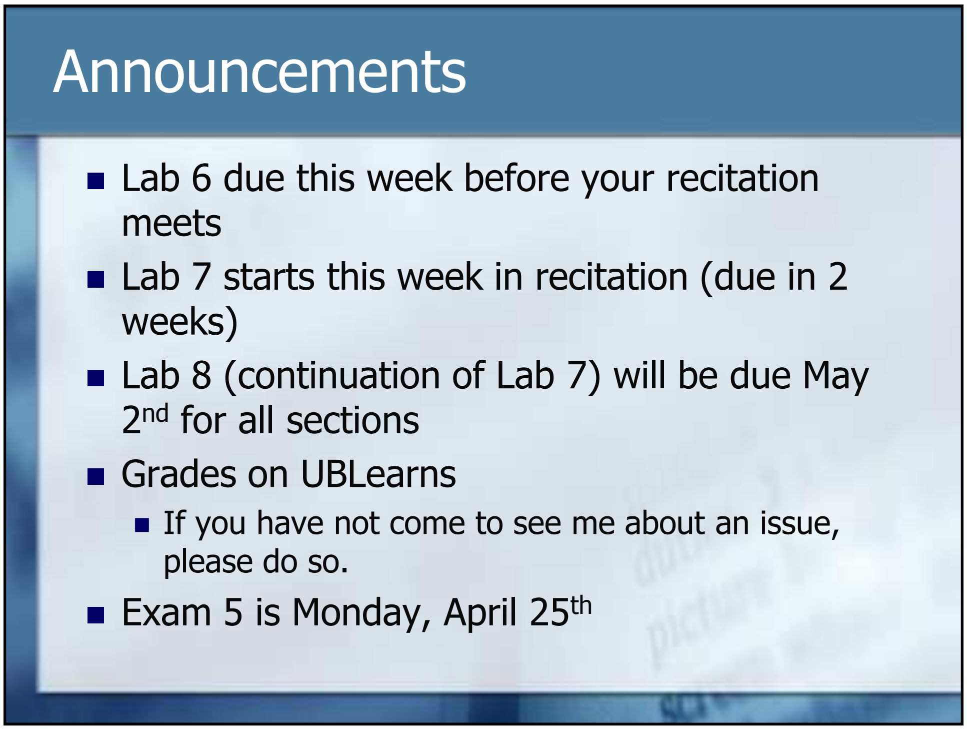


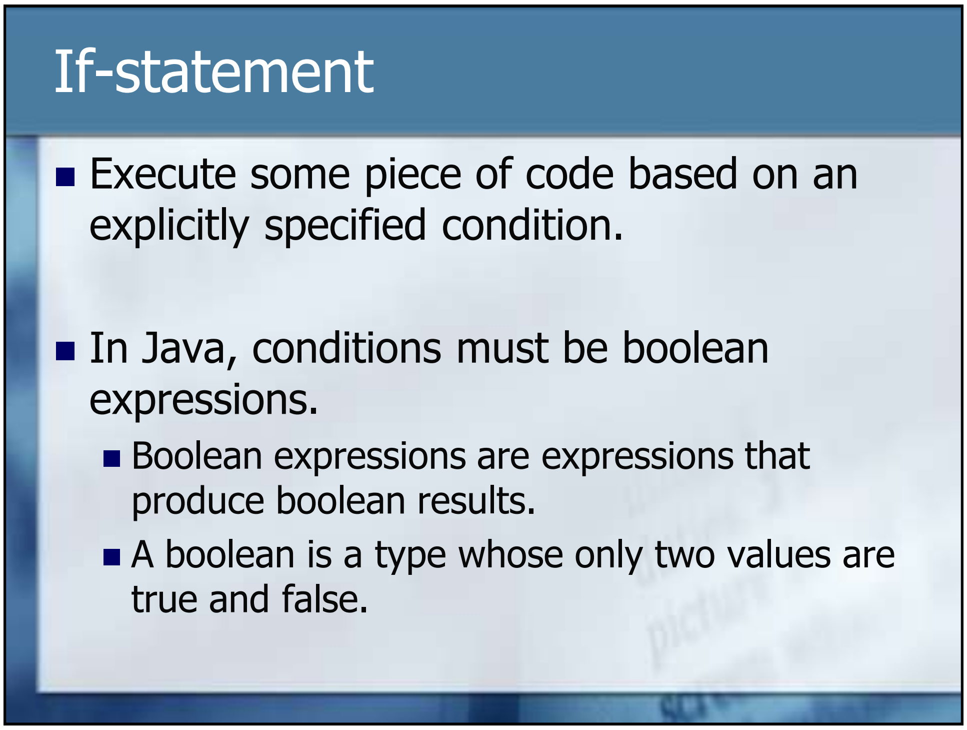
CSE 115/503

April 4 - 8, 2011



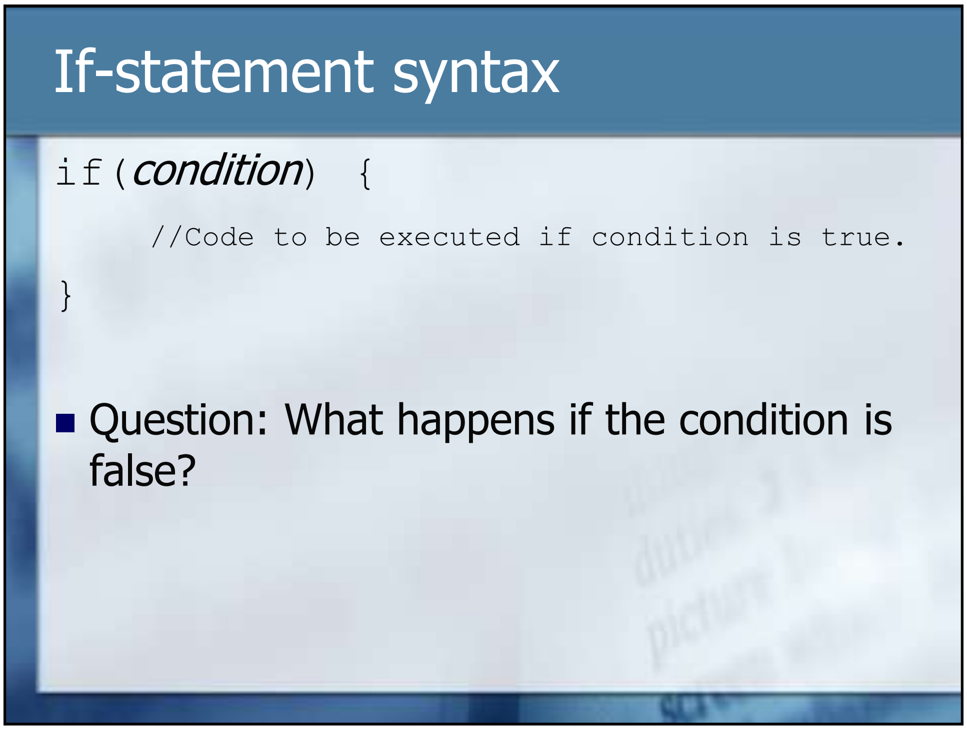
Announcements

- Lab 6 due this week before your recitation meets
- Lab 7 starts this week in recitation (due in 2 weeks)
- Lab 8 (continuation of Lab 7) will be due May 2nd for all sections
- Grades on UBLearns
  - If you have not come to see me about an issue, please do so.
- Exam 5 is Monday, April 25th



If-statement

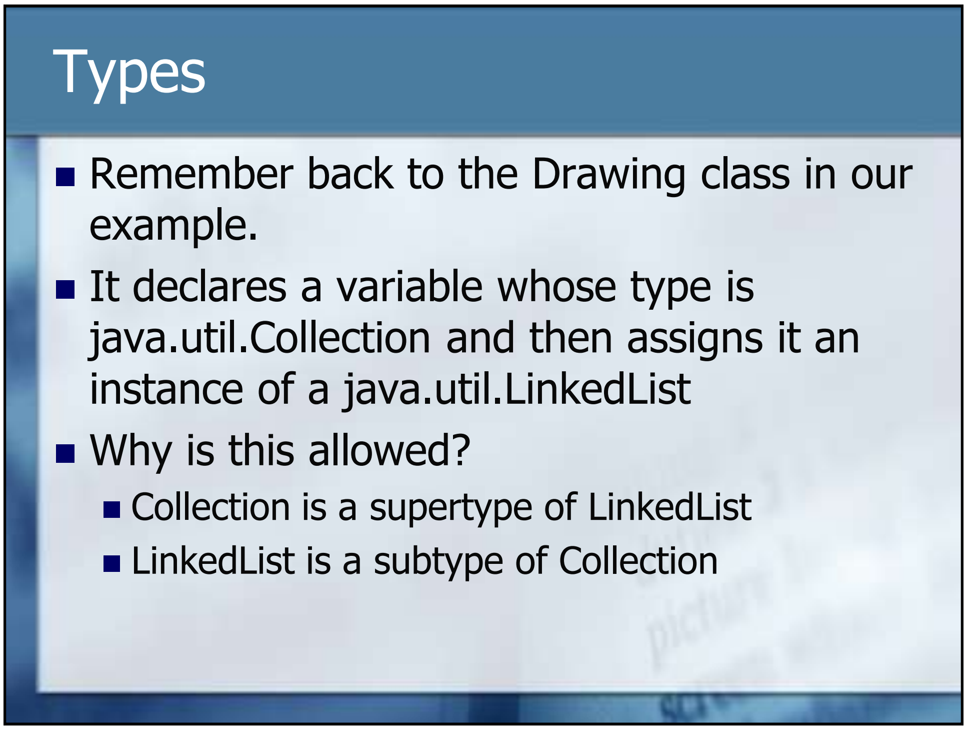
- Execute some piece of code based on an explicitly specified condition.
- In Java, conditions must be boolean expressions.
  - Boolean expressions are expressions that produce boolean results.
  - A boolean is a type whose only two values are true and false.



If-statement syntax

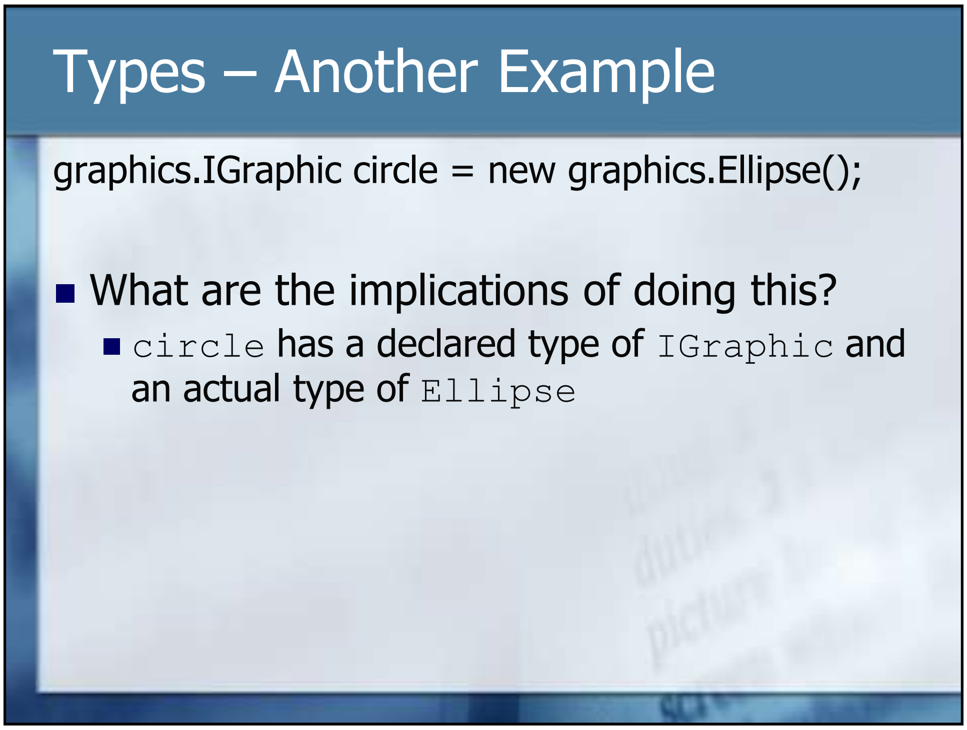
```
if(condition)  {
      //Code to be executed if condition is true.
}
```

- Question: What happens if the condition is false?



Types

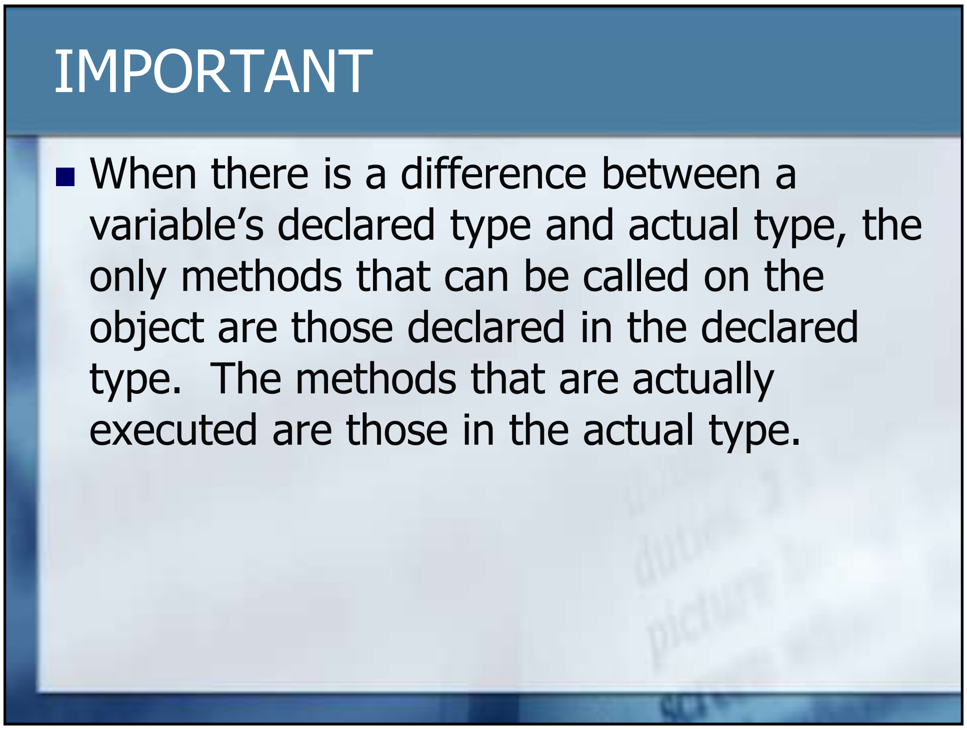
- Remember back to the Drawing class in our example.
- It declares a variable whose type is java.util.Collection and then assigns it an instance of a java.util.LinkedList
- Why is this allowed?
  - Collection is a supertype of LinkedList
  - LinkedList is a subtype of Collection



Types – Another Example

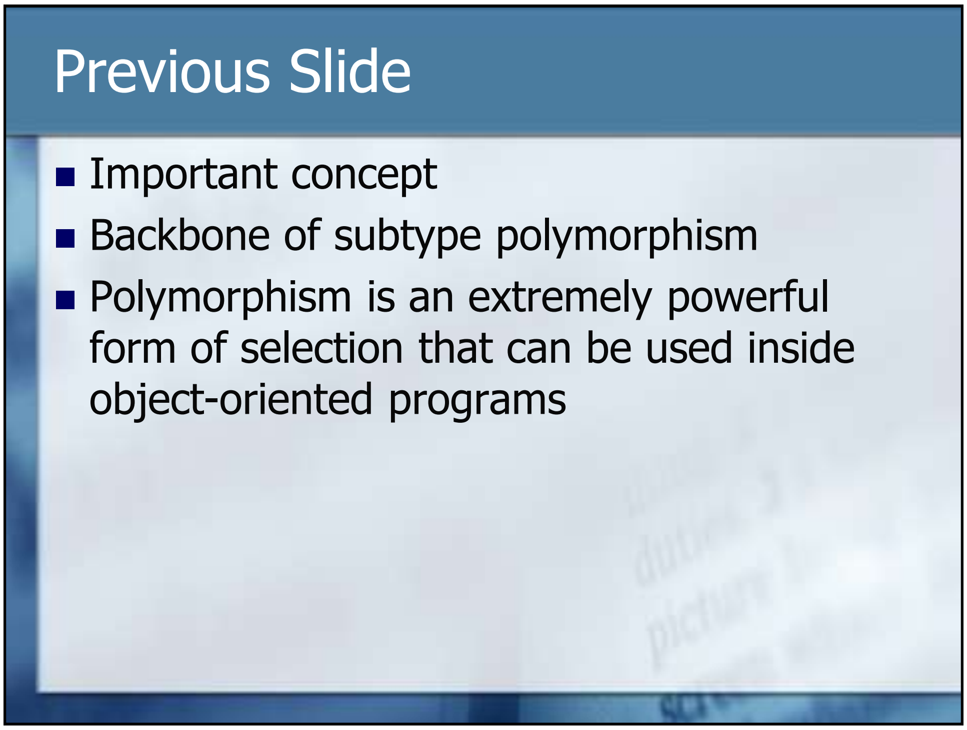
graphics.IGraphic circle = new graphics.Ellipse();

- What are the implications of doing this?
  - `circle` has a declared type of `IGraphic` and an actual type of `Ellipse`



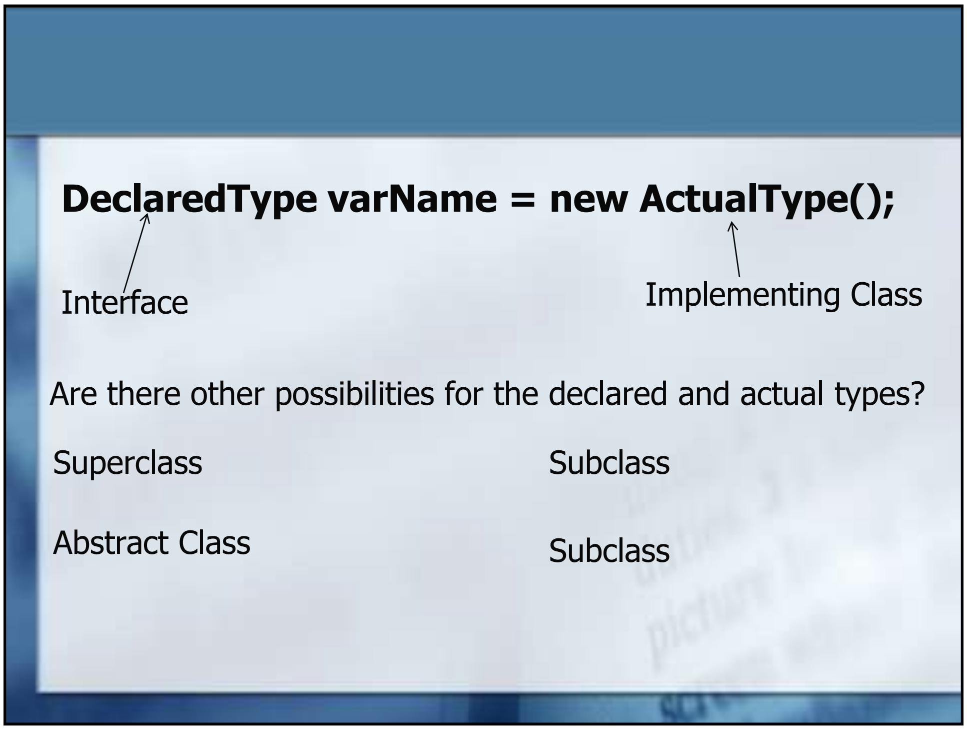
IMPORTANT

- When there is a difference between a variable’s declared type and actual type, the only methods that can be called on the object are those declared in the declared type. The methods that are actually executed are those in the actual type.



Previous Slide

- Important concept
- Backbone of subtype polymorphism
- Polymorphism is an extremely powerful form of selection that can be used inside object-oriented programs



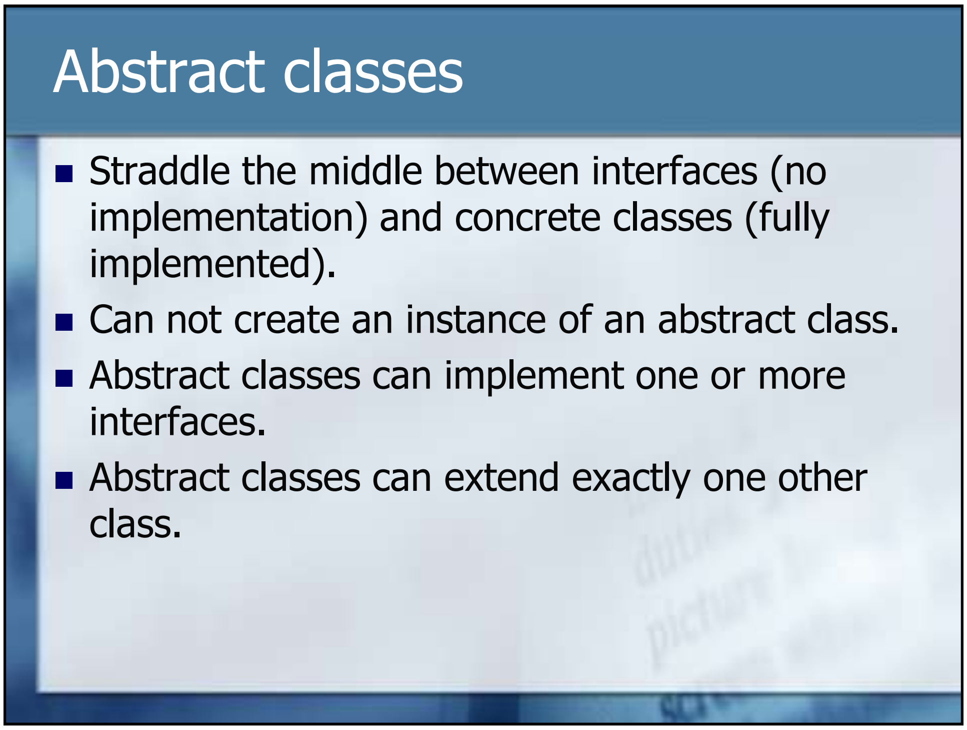
DeclaredType varName = new ActualType();

Interface

Implementing Class

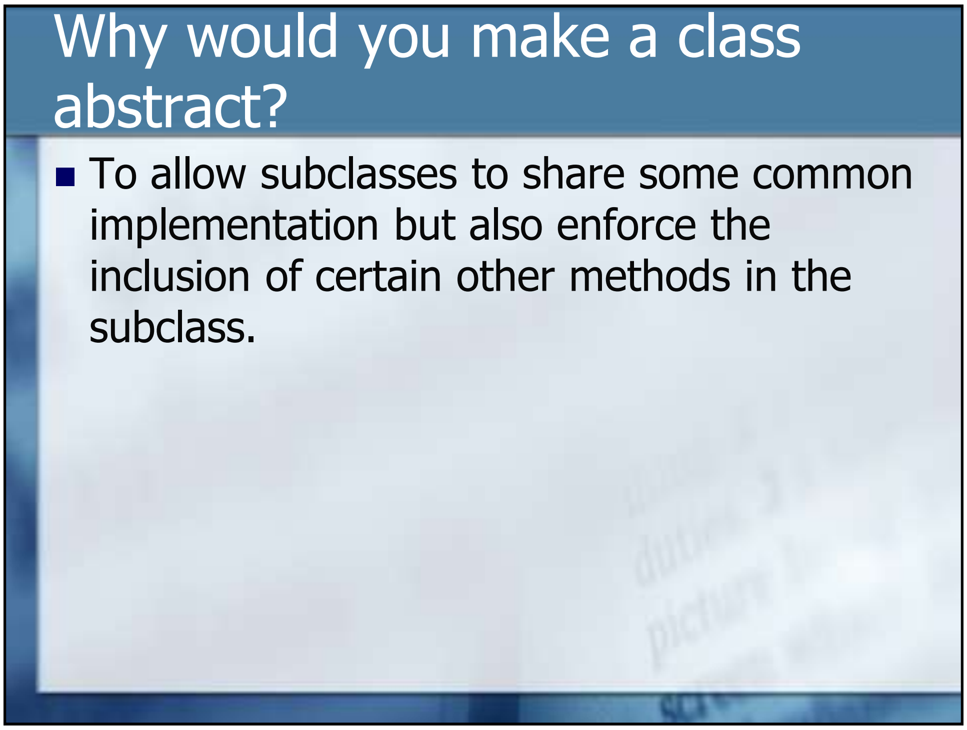
Are there other possibilities for the declared and actual types?

Superclass	Subclass
Abstract Class	Subclass



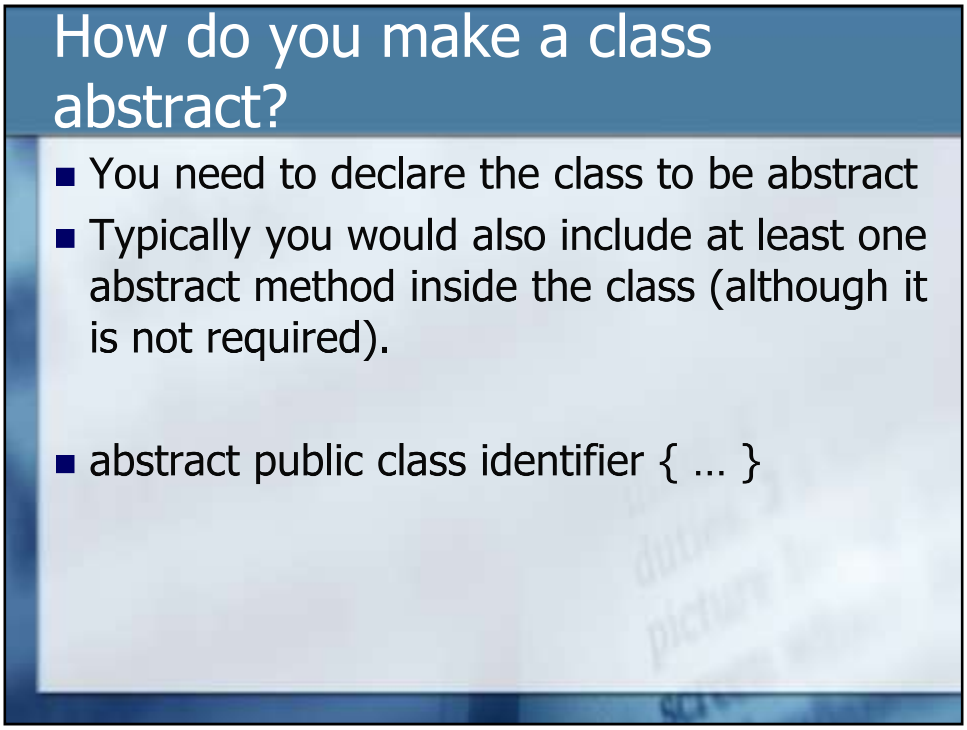
Abstract classes

- Straddle the middle between interfaces (no implementation) and concrete classes (fully implemented).
- Can not create an instance of an abstract class.
- Abstract classes can implement one or more interfaces.
- Abstract classes can extend exactly one other class.



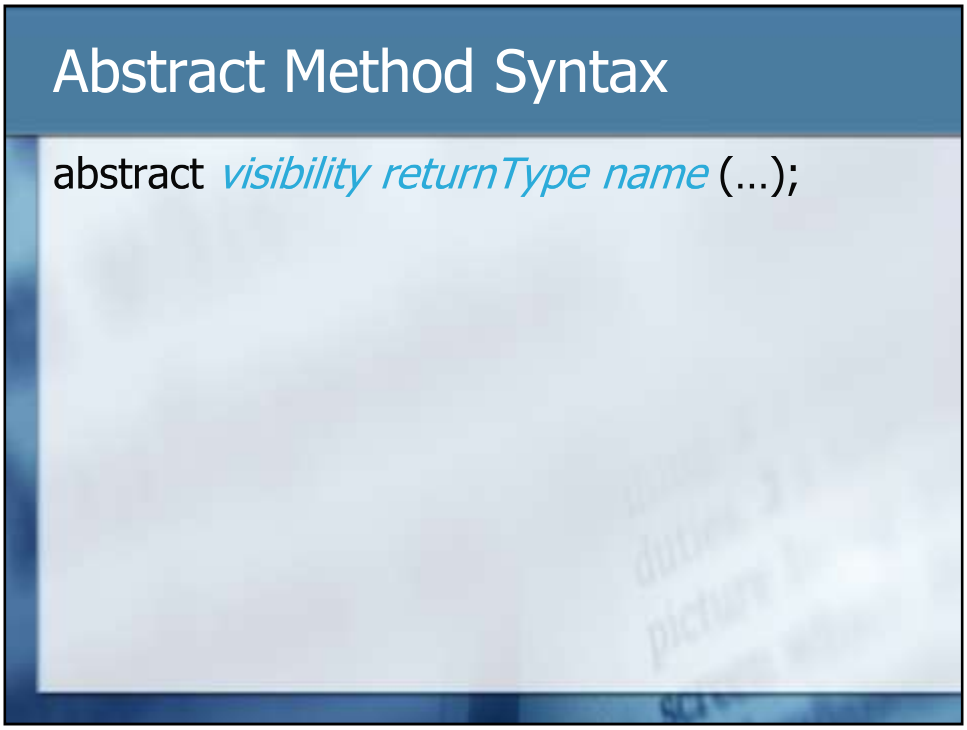
Why would you make a class abstract?

- To allow subclasses to share some common implementation but also enforce the inclusion of certain other methods in the subclass.



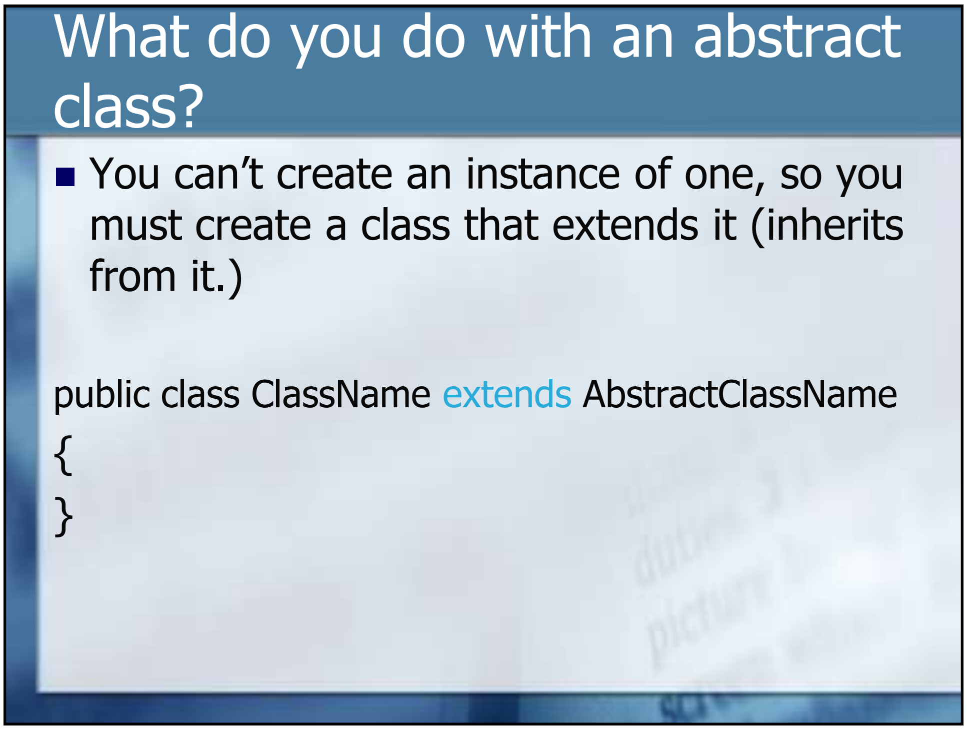
How do you make a class abstract?

- You need to declare the class to be abstract
- Typically you would also include at least one abstract method inside the class (although it is not required).
- abstract public class identifier { … }



Abstract Method Syntax

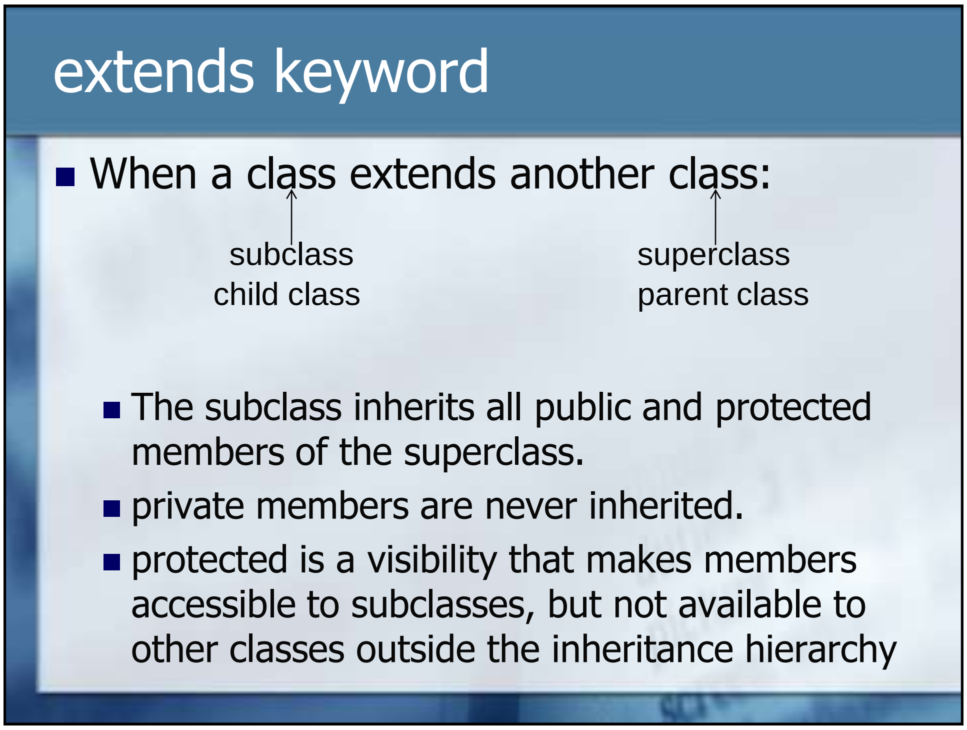
abstract *visibility returnType name* (…);



What do you do with an abstract class?

- You can't create an instance of one, so you must create a class that extends it (inherits from it.)

```
public class ClassName extends AbstractClassName
{
}
```



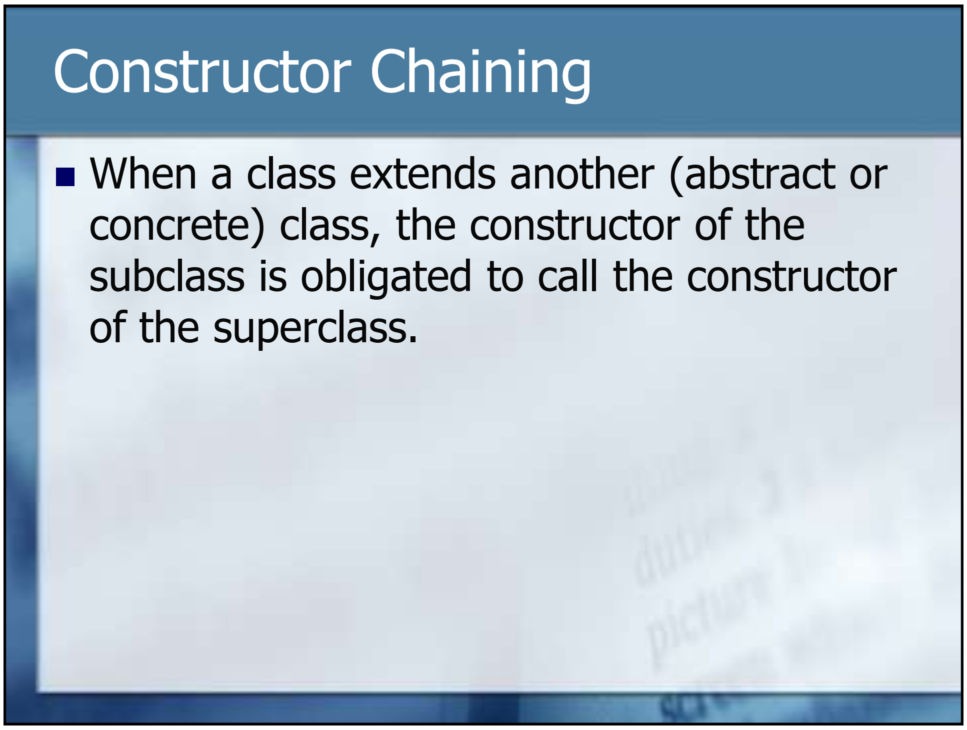
extends keyword

- When a class extends another class:

  subclass
  child class

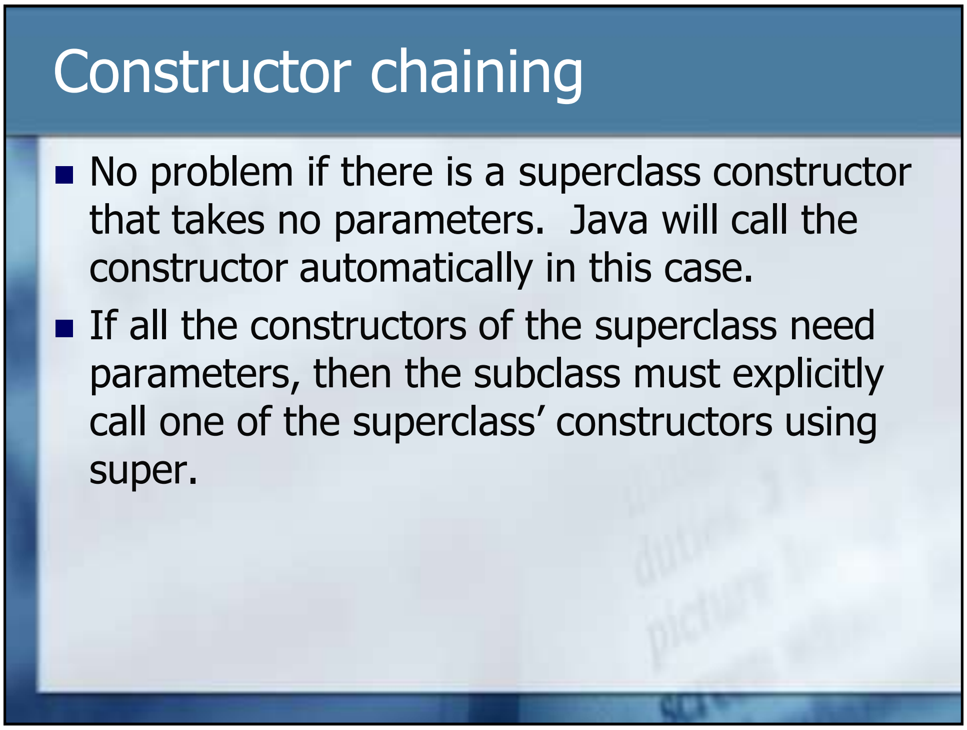
  superclass
  parent class

  - The subclass inherits all public and protected members of the superclass.
  - private members are never inherited.
  - protected is a visibility that makes members accessible to subclasses, but not available to other classes outside the inheritance hierarchy



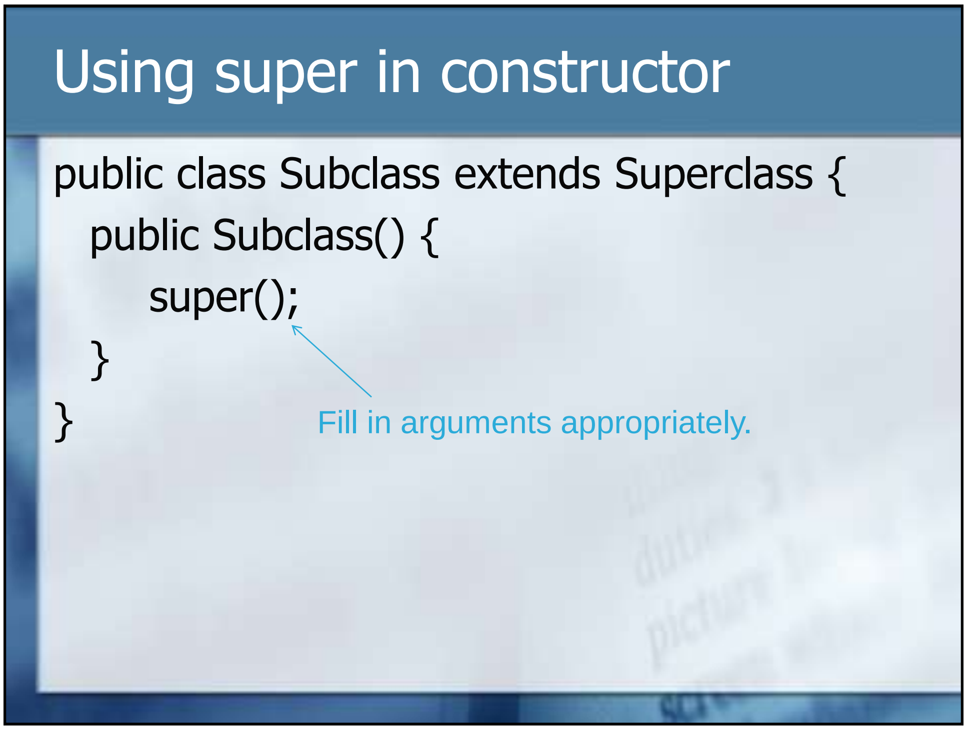
Constructor Chaining

- When a class extends another (abstract or concrete) class, the constructor of the subclass is obligated to call the constructor of the superclass.



Constructor chaining

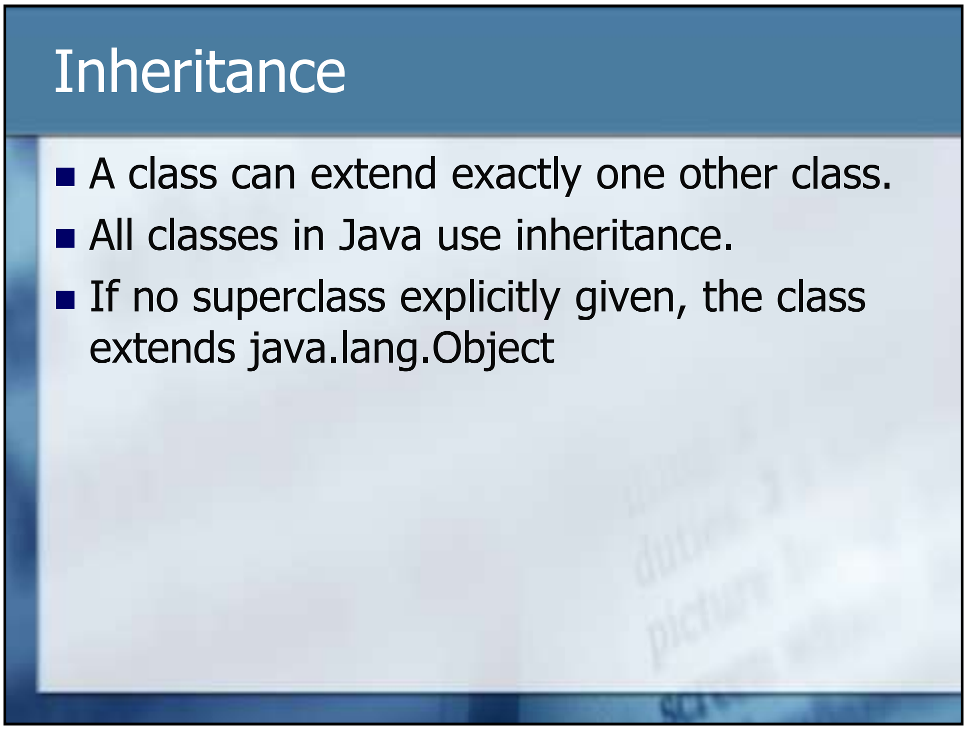
- No problem if there is a superclass constructor that takes no parameters. Java will call the constructor automatically in this case.
- If all the constructors of the superclass need parameters, then the subclass must explicitly call one of the superclass' constructors using super.



Using super in constructor

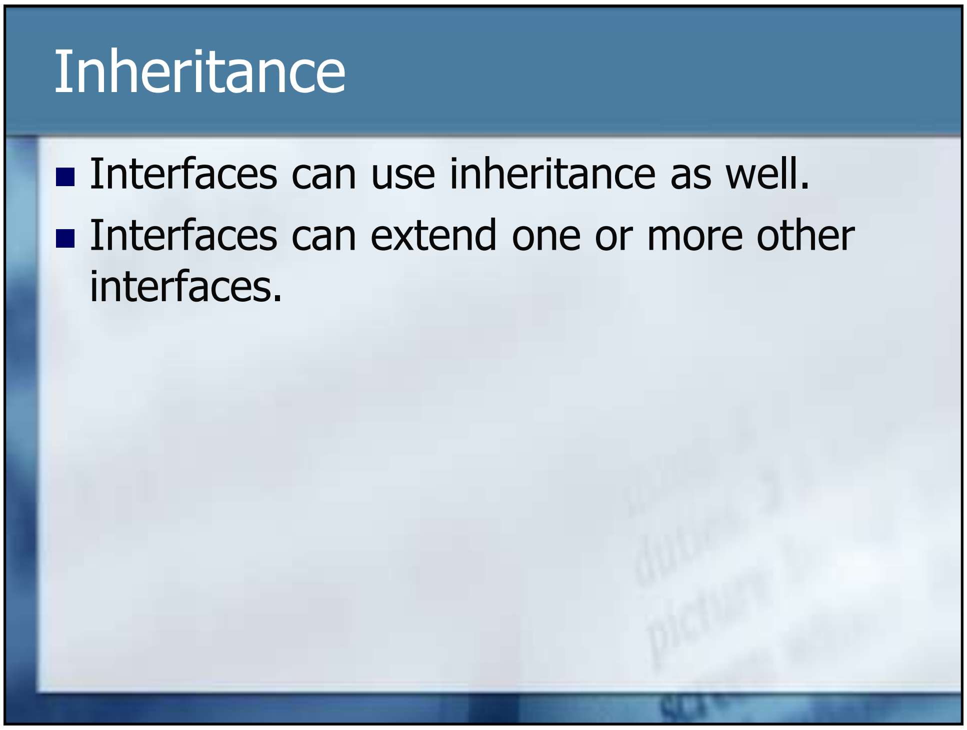
```
public class Subclass extends Superclass {
  public Subclass() {
      super();
  }
}
```

Fill in arguments appropriately.



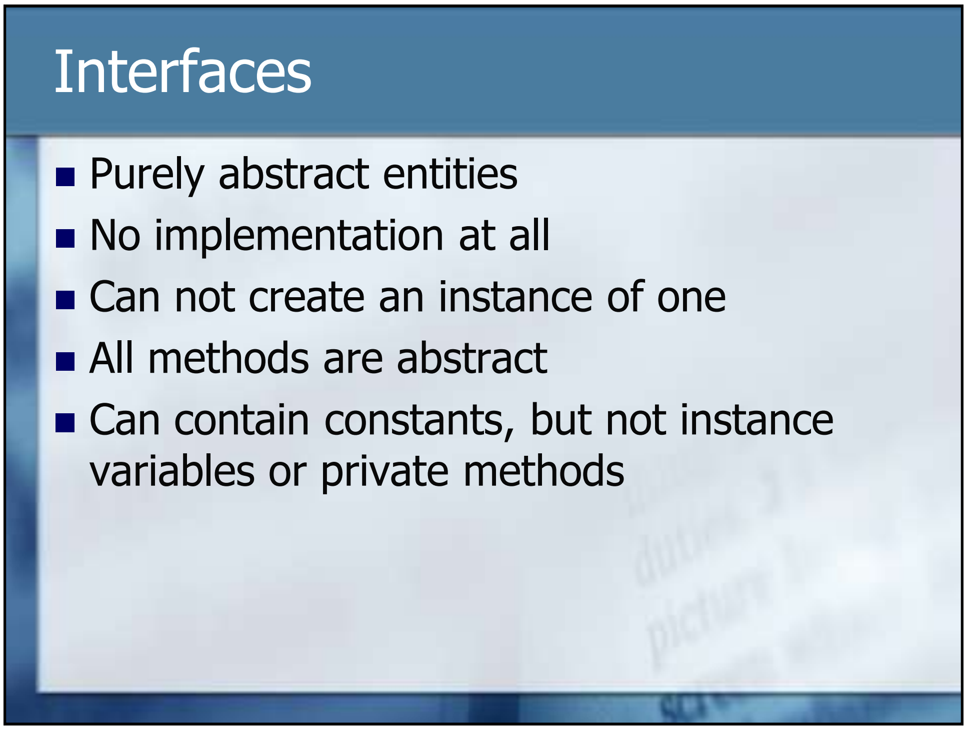
Inheritance

- A class can extend exactly one other class.
- All classes in Java use inheritance.
- If no superclass explicitly given, the class extends java.lang.Object



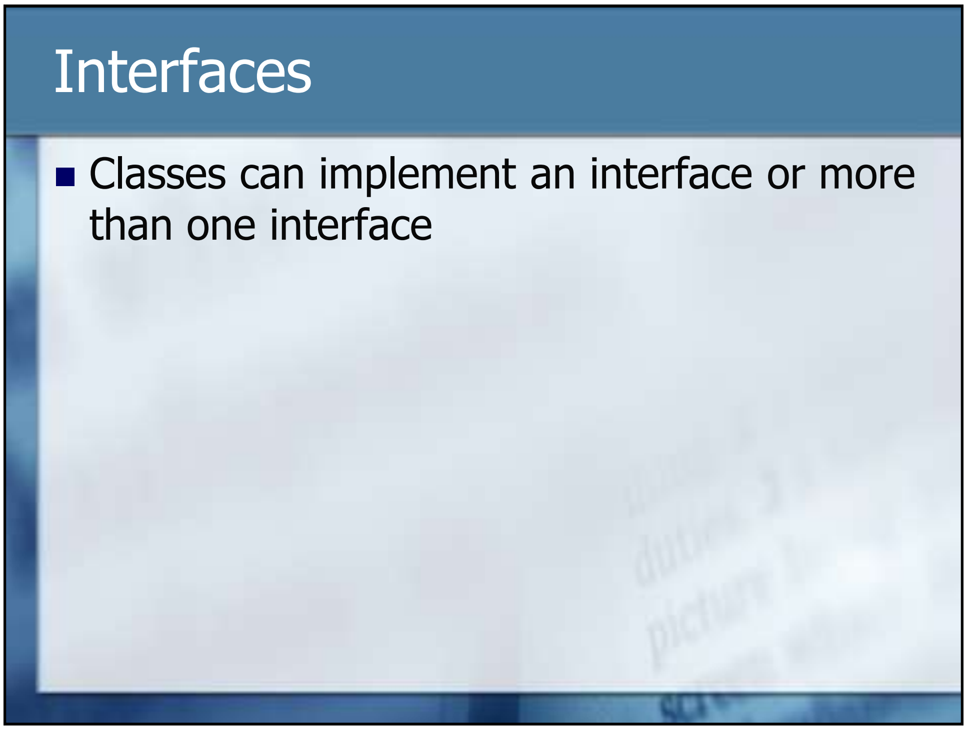
Inheritance

- Interfaces can use inheritance as well.
- Interfaces can extend one or more other interfaces.



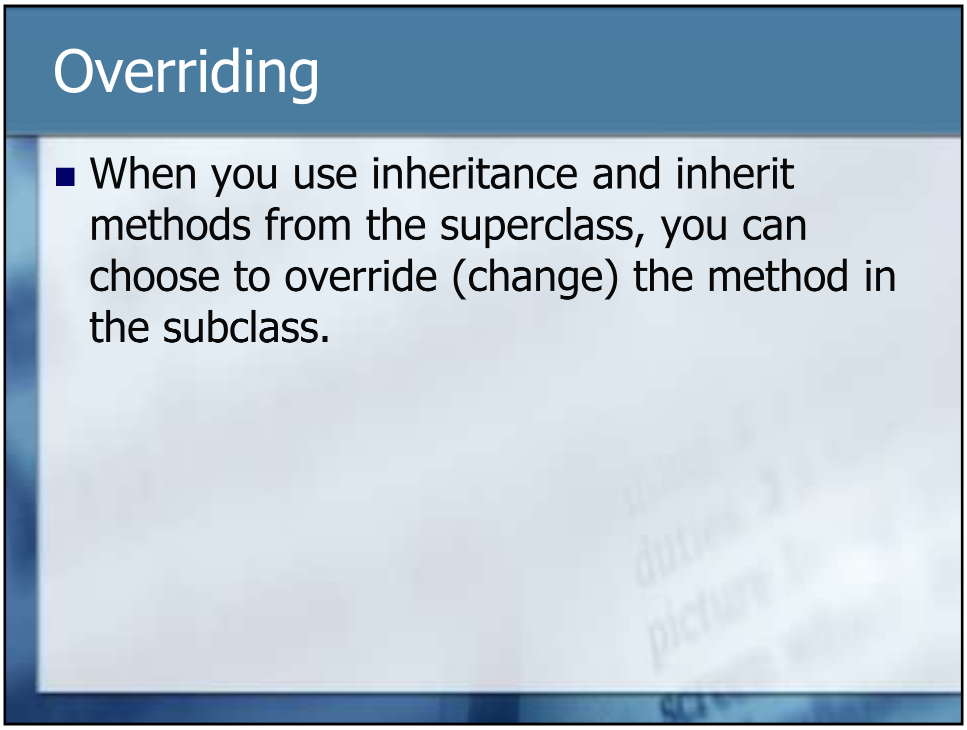
Interfaces

- Purely abstract entities
- No implementation at all
- Can not create an instance of one
- All methods are abstract
- Can contain constants, but not instance variables or private methods



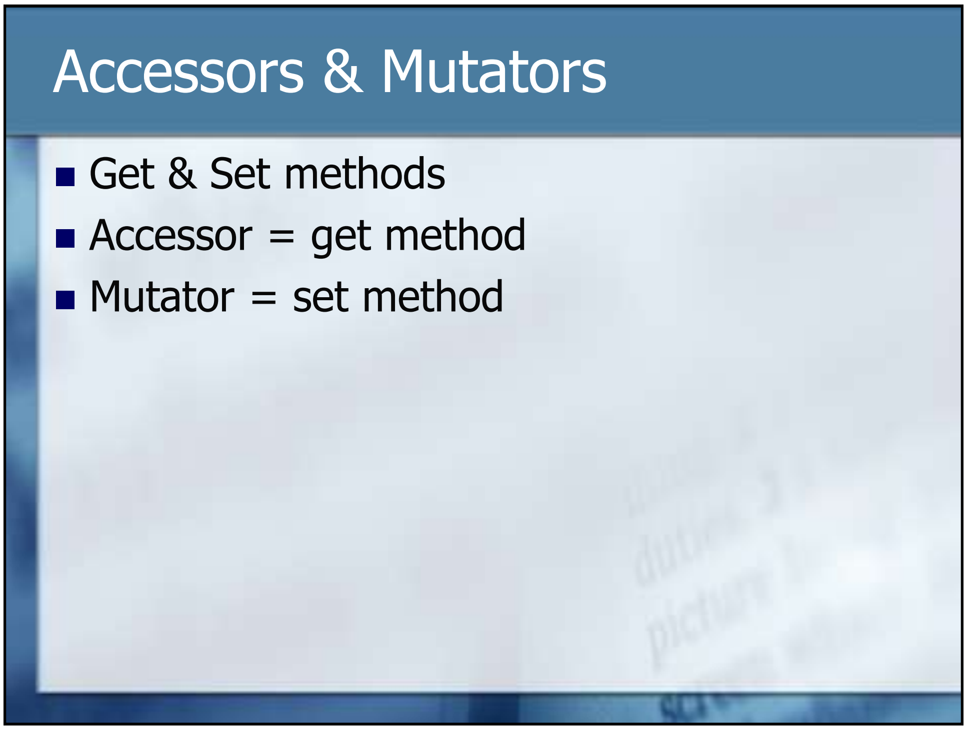
Interfaces

- Classes can implement an interface or more than one interface



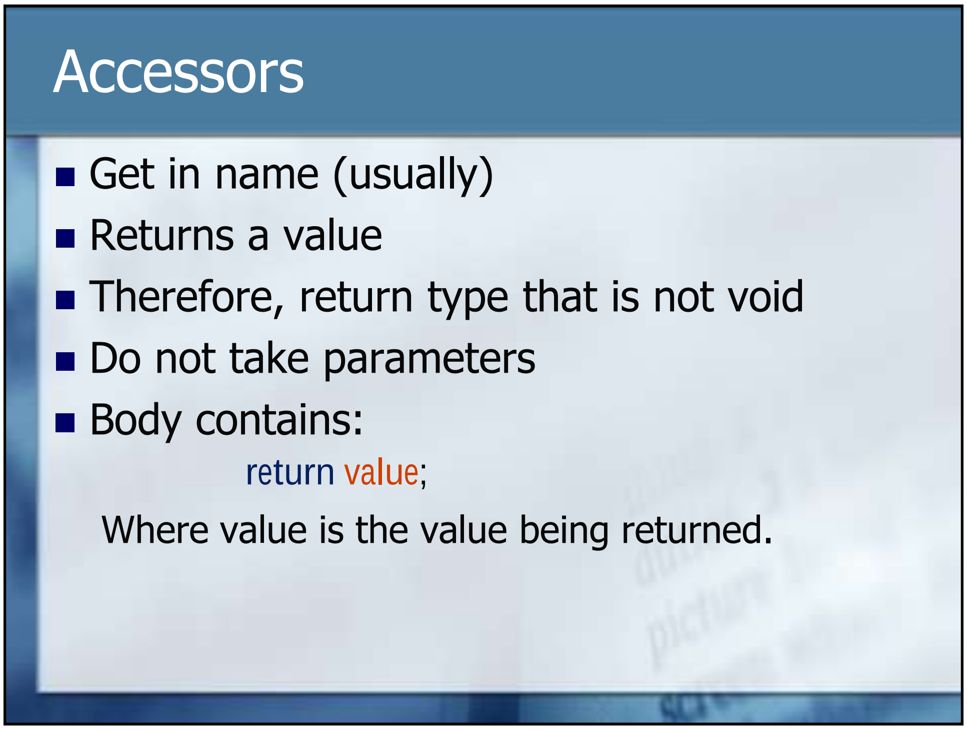
Overriding

- When you use inheritance and inherit methods from the superclass, you can choose to override (change) the method in the subclass.



Accessors & Mutators

- Get & Set methods
- Accessor = get method
- Mutator = set method

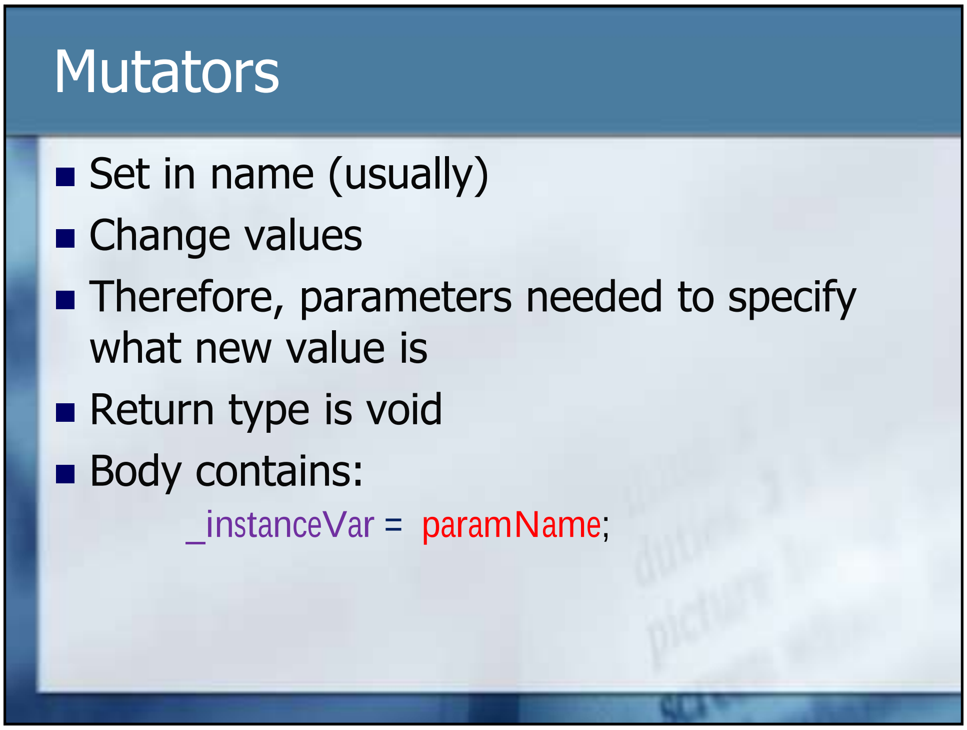


Accessors

- Get in name (usually)
- Returns a value
- Therefore, return type that is not void
- Do not take parameters
- Body contains:

```
return value;
```

Where value is the value being returned.



Mutators

- Set in name (usually)
- Change values
- Therefore, parameters needed to specify what new value is
- Return type is void
- Body contains:

```
_instanceVar = paramName;
```