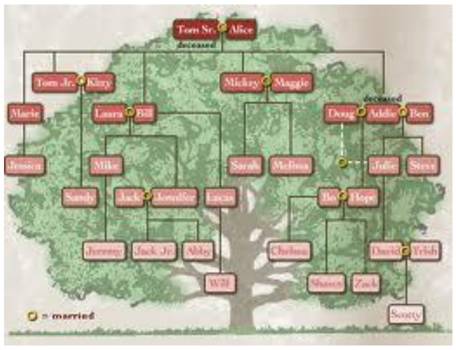
CSC115 Lecture 11

Making your data-type generic

- as oppposed to int, double, char, String, etc.
- Java's *generics*

Introducing:

- Inheritance →



Consider the Node class from Assignment 2

```
public class IntegerNode
{
    public IntegerNode    next;
    public IntegerNode    prev;
    public int          value;

    public IntegerNode()
    {
        //etc.
```

The value stored in the Node *must* be of type int

How to store double?

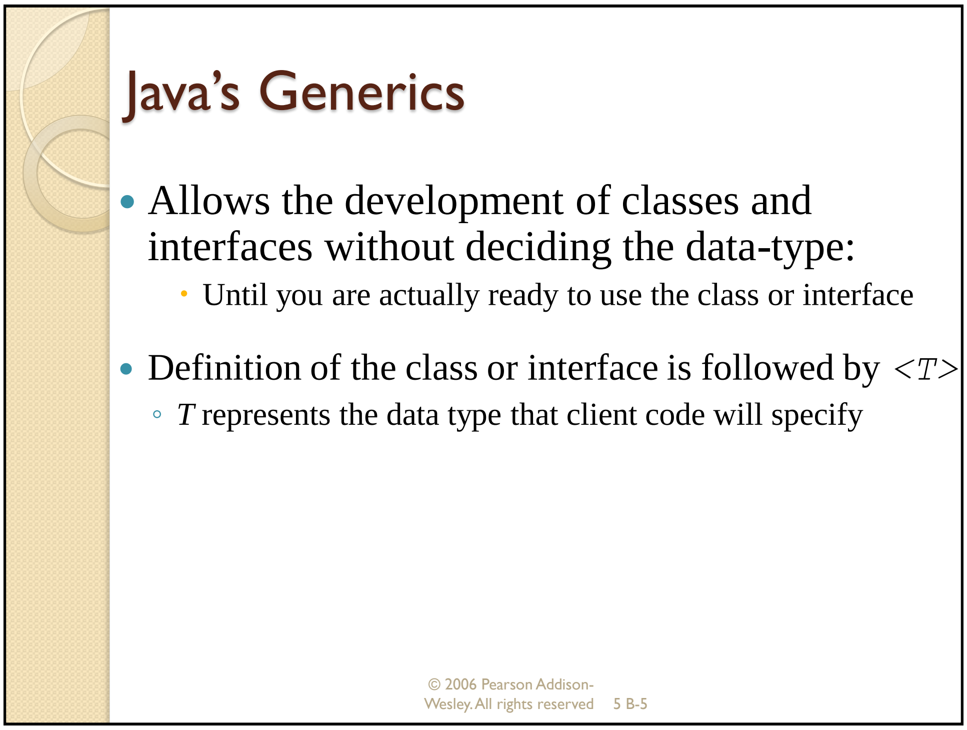
```
public class DoubleNode
{
    public DoubleNode    next;
    public DoubleNode    prev;
    public double        value;

    public DoubleNode()
    {
        //etc.
```

**But then . . .
What about:
char or String
or others?**

It does not make sense to re-write virtually the same code every time you want to change types. . . . .

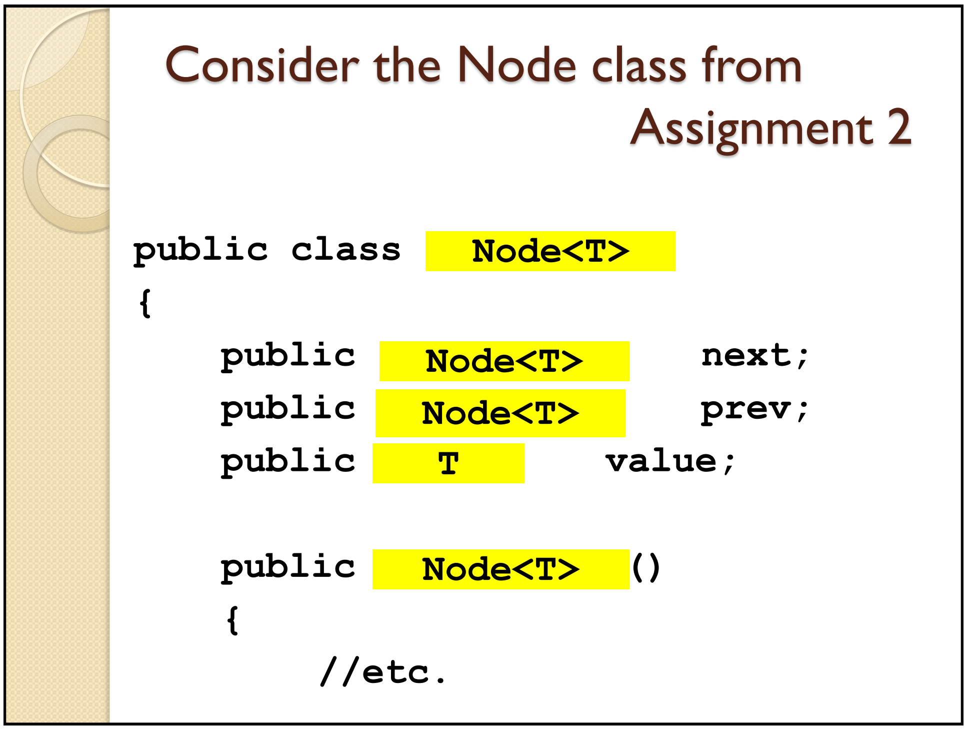
There must be something more generic!



Java's Generics

- Allows the development of classes and interfaces without deciding the data-type:
  - Until you are actually ready to use the class or interface
- Definition of the class or interface is followed by <*T*>
  - *T* represents the data type that client code will specify

© 2006 Pearson Addison-Wesley. All rights reserved



Consider the Node class from Assignment 2

```
public class  Node<T>
{
    public  Node<T>    next;
    public  Node<T>    prev;
    public  T       value;

    public  Node<T>  ()
    {
        //etc.
```

Review code in Lecture11Code
- Example with int
- Example with Generics

An issue with Generics and Arrays

- Try making an array of a class declared with arrays
- It gives an error!

- Alternative Solutions:
  - Use Java's ArrayList or Vector classes

```
Vector<T> test = new Vector<T>;
ArrayList<T> theTest = new ArrayList<T>;
```

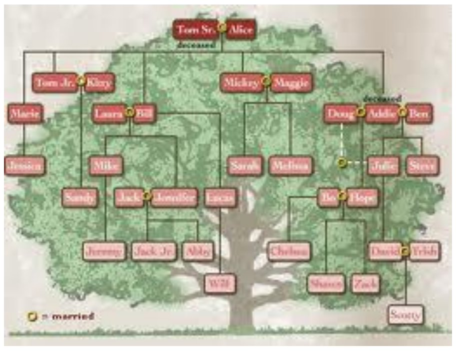
  - Use arrays of Objects with casting to T on output

    …Demo`d on another day!

Inheritance

Introducing:

- Inheritance →



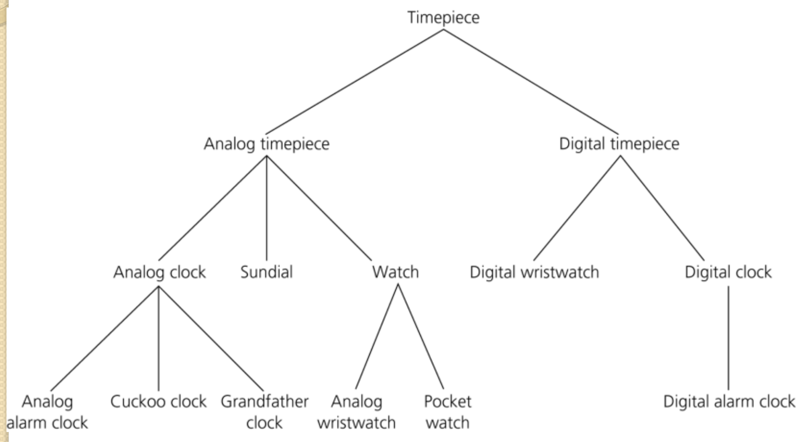
Some terms today

- Inheritance
- Super Class
- Sub Class
- Is-A
- Has-A

Watch for them…Try to write definitions for them , or distinguish the differences between them.

Inheritance

- Allows a class to derive the behavior and structure of an existing class



Inheritance - Terminology

- Superclass or base class
  - A class from which another class is derived
- Subclass, derived class, or descendant class
  - A class that inherits the members of another class
- Benefits of inheritance
  - It enables the reuse of existing classes
  - It reduces the effort necessary to add features to an existing object

Inheritance

- A subclass can
  - Add new members to those it inherits
  - Override an inherited method of its superclass
    - A method in a subclass overrides a method in the superclass if the two methods have the same declarations
    - Replacement: Provides a new implementation for the method
    - Refinement: Uses the superclass method as part of the subclass method.
      - <Examples next slide>

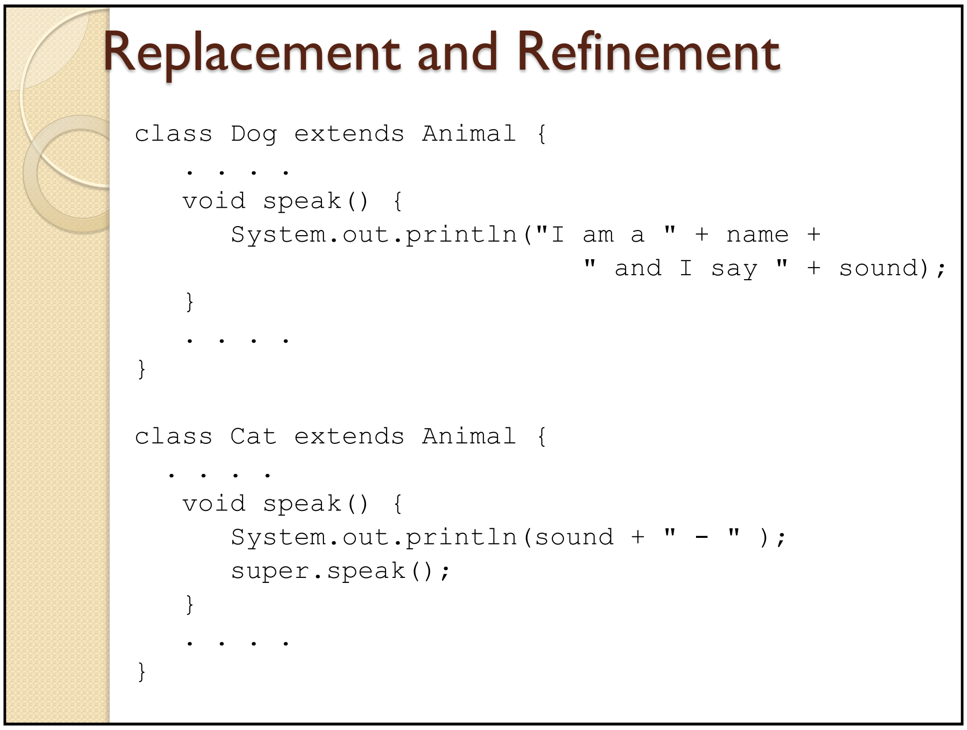
Constructors are automatically *refined*

A Base (Super) Class - Animal

```
class Animal {
  String className;
  String sound;

  public Animal () {
    this.className = " Mammalia :";
  }


   . . . .
   void speak() {
      System.out.print(this.className + " : " +
                                     this.sound );
   }
   . . . .
}
```

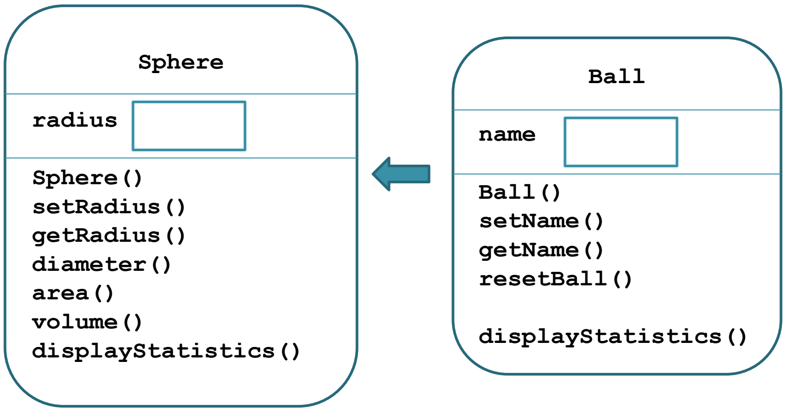


Replacement and Refinement

```
class Dog extends Animal {
   . . . .
   void speak() {
      System.out.println("I am a " + name +
                                 " and I say " + sound);
   }
   . . . .
}

class Cat extends Animal {
  . . . .
   void speak() {
      System.out.println(sound + " - " );
      super.speak();
   }
   . . . .
}
```

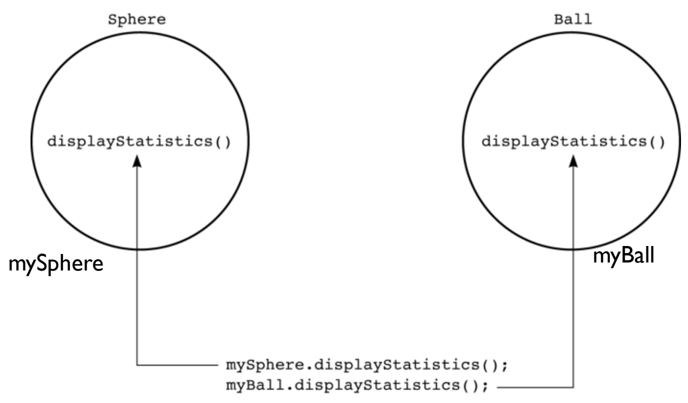
Inheritance
- Example Sphere & Ball



Inheritance

- A subclass inherits private members from the superclass, but cannot access them directly
- Methods of a subclass can call the superclass's public methods
- Clients of a subclass can invoke the superclass's public methods
- An overridden method
  - Instances of the subclass will use the new method
  - Instances of the superclass will use the original method

Inheritance



```
mySphere.displayStatistics();
myBall.displayStatistics();
```

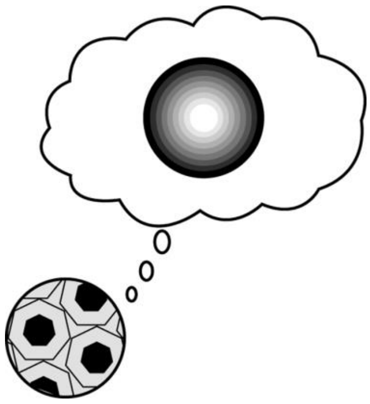
An object invokes the its own version of a method

Is-a and Has-a Relationships

- Two basic kinds of relationships
  - Is-a relationship
  - Has-a relationship

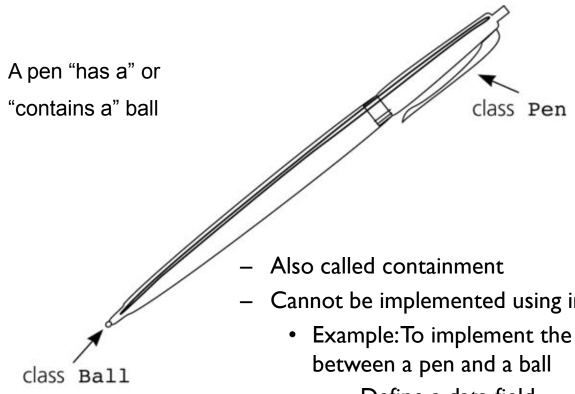
Is-a Relationship

- Inheritance should imply an is-a relationship between the superclass and the subclass
- Example:
  - If the class `Ball` is derived from the class `Sphere`
    - A ball is a sphere



- Compatibility: An instance of a subclass can be used instead of an instance of the superclass, but not the other way around

Has-a Relationships

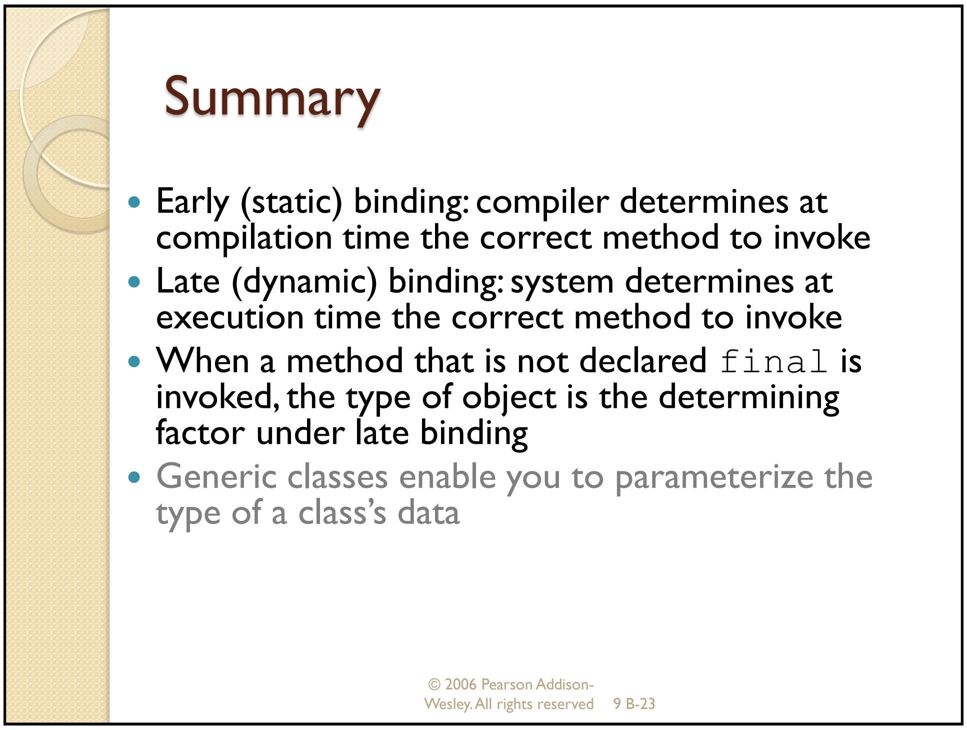


- Also called containment
- Cannot be implemented using inheritance
  - Example: To implement the has-a relationship between a pen and a ball
    - Define a data field `point` – whose type is `Ball` – within the class `Pen`

Summary

- A subclass inherits all members of its previously defined superclass, but can access only the public and protected members
- Subclasses and superclasses
  - A subclass is type-compatible with its superclass
  - The relationship between superclasses and subclasses is an is-a relationship
- A method in a subclass overrides a method in the superclass if they have the same parameter declarations

© 2006 Pearson Addison-Wesley. All rights reserved



Summary

- Early (static) binding: compiler determines at compilation time the correct method to invoke
- Late (dynamic) binding: system determines at execution time the correct method to invoke
- When a method that is not declared `final` is invoked, the type of object is the determining factor under late binding
- Generic classes enable you to parameterize the type of a class's data

© 2006 Pearson Addison-Wesley. All rights reserved