Lecture 10

• Topics Covered
  – Inheritance
    • Abstract Classes
    • Upcasting & Downcasting
    • Polymorphism
Polymorphism Recap

- Polymorphism
  - In English: Many forms, or many meanings
  - In OOP: One method name, multiple implementations
- 3 different types of polymorphism
  - Method overloading
  - Upcasting
  - Dynamic or Late Binding
Dynamic Binding

• Binding is the process of deciding what method is to be called
  – This is typically decided by the name and parameter list at compile time
    • This is usually known as static binding
  – When methods of “is-a” related classes have the same name and methods (methods are overridden), the decision must be made at runtime (binding is late... dynamic)
Abstract Classes

- Some classes exist solely to provide an inheritance hierarchy so
  - Code can be written once and reused in similar classes
  - Lists of similar data can be maintained

- An abstract class cannot be instantiated

- To make a class abstract, we add the reserved word `abstract` to the class header
  - `public abstract class ClassName{...}`
Abstract methods

• Abstract classes may contain abstract methods
  – An abstract method has no implementation (just use a stub for the method prototype)
  – public abstract void methodName();

• If any derived class extended from an abstract class with an abstract method does not implement the abstract method, the derived class is automatically abstract as well
Upcasting and Downcasting

- Upcasting allows a base class reference to refer to a derived class object.
  - Ex: Shape sh = new Triangle();
  - Triangle “is-a” Shape, so a Shape reference can refer to a Triangle object
  - We can call all methods that exist in Shape or are overridden in Triangle with sh
    - Ex: sh.CalcArea();
    - The appropriate CalcArea() will be called based upon the object sh refers to
Issue with Upcasting

• Let’s assume, we have created a new method in Triangle that we want to call that does not exist in the Shape class, what do you think about the following statements?
  – Shape sh = new Triangle();
  – int c = sh.GetThirdSide(a,b);

• The compiler will complain since the compiler makes the decision based upon the type of the reference at compile time
  – In order to fix this, we have to explicitly downcast
    • Casting an object to a derived or more specialized type
  – Shape sh = new Triangle();
  – int c = ((Triangle)sh).GetThirdSide(a,b);
instanceof operator

- **Syntax**
  - obj instanceof className
    - Returns true if obj “is-a” className
    - Return false if obj is not a className
- **Can be helpful in deciding when to cast a base class reference**
  - if (shapelist[1] instanceof Circle)
    double circ = ((Circle)shapelist[1]).GetCircumference();
Java Language and Inheritance

- The Java Language actually benefits from Inheritance
- Every class that is ever created in Java extends the Object class
  - If a method has an Object data type as a parameter, any class Name will have the “is-a” property with Object and can be passed in as a parameter
  - The Object type allows us to generalize methods
- The Object class is the base class of all classes
  - Every class inherits the Object methods
    - public boolean equals(Object object)
    - public String toString()
equals and toString

- When we use == with non-primitive objects, what is compared?
  - The equals method in the Object class does the same thing; however, we can override the method to check actual values

- Likewise, we can override the toString method to return a String as we wish