CMSC 132:
Object-Oriented Programming II

Abstract Classes

Department of Computer Science
University of Maryland, College Park
Modifier – Abstract

**Description**
- Represents generic concept
- Just a placeholder
- Leave lower-level details to subclass

**Applied to**
- Methods
- Classes

**Example**

```java
abstract class Foo {
    // abstract class
    abstract void bar( ) { ... } // abstract method
```
Abstract – Motivating Example

Graphics drawing program

- Define a base class Shape
- Derive various subclasses for specific shapes
- Each subclass defines its own method `drawMe()`

```java
public class Shape {
    public void drawMe() { ... }  // generic drawing method
}
public class Circle extends Shape {
    public void drawMe() { ... }  // draws a Circle
}
public class Rectangle extends Shape {
    public void drawMe() { ... }  // draws a Rectangle
}
```
Motivating Example – Shapes

**Implementation**

- **Picture consists of array shapes of type Shape[ ]**
- **To draw the picture, invoke drawMe( ) for all shapes**

```java
Shape[ ] shapes = new Shape[...];
shapes[0] = new Circle( ... );
shapes[1] = new Rectangle( ... );
...
for ( int i = 0; i < shapes.length; i++ )
    shapes[i].drawMe( );
```

Store the shapes to be drawn in an array.

Heap:

- [0]: (a Circle object)
- [1]: (a Rectangle object)
- [2]: ...
- ...
Motivating Example – Shapes

Problem

- **Shape** object does not represent a specific shape
  - Since Shape is just a superclass

How to implement Shape’s `drawMe( )` method?

```java
public class Shape {
    void drawMe() { ... } // generic drawing method
}
```
Motivating Example – Shapes

Possible solutions

- Draw some special “undefined shape”
- Ignore the operation
- Issue an error message
- Throw an exception

Better solution

- Abstract drawMe( ) method, abstract Shape class
- Tells compiler Shape is incomplete class
Abstract Method

- Behaves much like method in interface
- Give a signature, but no body
- Includes modifier `abstract` in method signature
- Class descendants provide the implementation
- Abstract methods cannot be final
  - Since must be overridden by descendant class
  - Final would prevent this
Abstract Class

- Required if class contains any abstract method
- Includes modifier `abstract` in the class heading
  
  ```java
  public abstract class Shape { … }
  ```

- An abstract class is incomplete
  
  - Cannot be created using “new”
    ```java
    Shape s = new Shape( … ); // Illegal!
    ```
  
  - But can create concrete shapes (Circle, Rectangle) and assign them to variables of type Shape
    ```java
    Shape s = new Circle( … );
    ```
public abstract class Shape {
    private int color;
    Shape ( int c ) { color = c; }
    public abstract void drawMe( );
}

public class Circle extends Shape {
    private double radius;
    public Circle( int c, double r ) { ... details omitted ... }
    public void drawMe( ) { ... Circle drawing code goes here ... }
}

public class Rectangle extends Shape {
    private double height;
    private double width;
    public Rectangle( int c, double h, double w ) { ... details omitted ... }
    public void drawMe( ) { ... Rectangle drawing code goes here ... }
}

Base class Shape is abstract because it contains the abstract (undefined) method drawMe( ).

Derived class Circle is concrete because it defines drawMe( ).

Derived class Rectangle is concrete because it defines drawMe( ).

The code for drawing the shapes given earlier can now be applied.
Abstract – Summary

Abstract methods

- Method that contains no body
- Subclass provides actual implementation

Abstract classes

- Required if any method in class is abstract
- Can contain non-abstract methods
- Can be partial description of class