CMSC 132:
OBJECT-ORIENTED PROGRAMMING II

Abstract Classes/Modifiers

Department of Computer Science
University of Maryland, College Park
Motivating Example – Shapes

- **Example:** AbsClassesModifiersCode
- **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.
    Draws all the shapes. Each call invokes drawMe for the specific shape.

Heap:

```
[0]                    [1]                    [2]                    ...
(a Circle object)      (a Rectangle object) ...
```

shapes

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

• Graphics drawing program
  • Define a base class Shape
  • Derive various subclasses for specific shapes
  • Each subclass defines its own method drawMe()

```
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
}
```

• If we only need the drawMe() method, could we have used an interface?
• Notice we want to place common methods in based class (in addition to have drawMe()
Motivating Example – Shapes

• Problem
  • `Shape` object does not represent a specific shape, still users can create instances of it (`Shape s = new Shape();`)

  • How to implement Shape’s `drawMe()` method?
    ```java
    public class Shape {
        void drawMe() { … }    // generic drawing method
    }
    ```

• 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
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
  }
  
  Example: AbsClassesModifiersCode
  ```
Abstract Class

- **Abstract Methods**
  - 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” → `Shape s = new Shape( … ); // Illegal!`
    - But can create concrete shapes (Circle, Rectangle) and assign them to variables of type Shape → `Shape s = new Circle( … );`
Modifiers

- Description
  - Java keyword (added to definition)
  - Specifies characteristics of a language construct
- (Partial) list of modifiers
  - Visibility modifiers (public / private / protected)
  - static
  - final
  - abstract
Visibility Modifiers

- **public**
  - Referenced anywhere (i.e., outside package)

- **private**
  - Referenced only within class definition
  - Applicable to class fields & methods

- **protected**
  - Referenced within package, or by subclasses outside package

- **None specified (package)**
  - Referenced only within package
Visibility Modifier

```java
package fooBar;
public class A {
    public int vPub;
    protected int vProt;
    int vPack;
    private int vPriv;
}
```

```java
package fooBar;
public class B {
    can access vPub;
    can access vProt;
    can access vPack;
    cannot access vPriv;
}
```

```java
package fooBar;
public class C extends A {
    can access vPub;
    can access vProt;
    can access vPack;
    cannot access vPriv;
}
```

```java
public class D extends A {
    can access vPub;
    can access vProt;
    cannot access vPack;
    cannot access vPriv;
}
```

```java
public class E {
    can access vPub;
    cannot access vProt;
    cannot access vPack;
    cannot access vPriv;
}
```
Static Modifier

• Static variable
  • Single copy for class
  • Shared among all objects of class
• Static method
  • Can be invoked through class name
  • Does not need to be invoked through object
  • Can be used even if no objects of class exist
  • Can not reference instance variables
• Example: AbsClassesModifiersCode
Final Modifier

- **Final variable**
  - Value can not be changed
  - Must be initialized in every constructor
  - Attempts to modify final are caught at compile time
- **Final static variable**
  - Used for constants
  - Example
    ```java
    final static int Increment = 5;
    ```
- **Final method**
  - Method **can not be overridden** by subclass
  - Private methods are implicitly final
- **Example**: AbsClassesModifiersCode
- **Final class**
  - Class can not be a superclass (extended)
  - Methods in final class are implicitly final
  - Prevents inheritance / polymorphism
  - May be useful for
    - Security
    - Object oriented design
  - **Example**: String class