1 **Objected-Oriented Programming: Abstract Classes and Interfaces**

CST242

2 **Polymorphism (Page 1)**

- Programmers should create systems that are easily extensible
  - Easy to add to later—capable of being *extended*
- Superclasses are designed as more general:
  - Able to process all *existing* and *new* subclasses
  - Classes that are added later will not require modification to the general part of the program (its superclass)

3 **Polymorphism (Page 2)**

- Late binding—a method from one class is not tied to method that calls it from another class until run-time (when it is instantiated)
  - Also called dynamic binding
  - The opposite of early binding in which the two methods are *compiled* together
- Late binding makes it possible to add new classes to the hierarchy even after the base class compiles

5 **Polymorphism (Page 3)**

- Consider the Shape class example:
  - Shape has:
    - An attribute named point where shape starts to draw
    - A method named center() that centers the shape when drawn by calling a method named position()
  - Classes Circle and Rectangle both extend Shape
    - Circle has attribute radius; Rectangle has attributes length and width
    - Circle and radius have individual methods named draw() that draw the shapes, both of which are *called* by the center() method of Shape

7 **Polymorphism (Page 4)**

- Consider the Shape class example (*con*):
  - With early binding, if *new* class Triangle is created after Shape is compiled, method draw() of either Circle or Rectangle will have been bound previously to center()
  - With late binding (essentially the *equivalent of polymorphism*), method draw() of Triangle (or Circle or Rectangle) correctly binds to center() at *run-time*
  - Java uses late binding exclusively

9 **The Keyword abstract (Page 1)**

- Classes that are declared to be abstract cannot be instantiated …
  - No objects may be created from it
- This is true for a superclass that only has the function of *supporting subclasses* …
  - Such classes are called abstract superclasses

10 **The Keyword abstract (Page 2)**

- Example:
  private abstract class Shape extends Object
- Classes that *may* instantiate objects are called concrete classes
  – E.g. the Circle, Rectangle and Triangle classes

11 **Declaring abstract Methods (Page 1)**

- A method may be declared in a superclass declaration as abstract
- As such the abstract method only may exist in an abstract class (or an interface)

12 **Declaring abstract Methods (Page 2)**

- The declaration is only a *reference* since:
  – It contains *no statements*
  – Requires implementation of the abstract method in all of its subclasses (so that the required methods are not forgotten in the subclasses)
  – Any call to the local abstract method is *overridden* because it will be handled by methods of same name in the subclasses (uses redirection)
  – In fact this is the only way that a superclass can *call methods of its direct subclass*

13 **Declaring abstract Methods (Page 3)**

- Format:
  public abstract *type*/void *methodName*([*parameterList*]);
  – The *parameterList* must match in number of variables and type the implemented method
  – Methods that are abstract may be *overloaded*
- Example:
  public abstract void draw();
  – Note the placement of the semicolon (;) at end of the method header (signature)

16 **The Keyword final (Review)**

- Used to indicate that value of an identifier *may not change* after it has been declared and initialized
  – Often used for defining a constant
- Example:
  double final CREDITS = 7;

17 **Declaring a Class as final**

- If a class is declared to be final, it must be the bottom class in an inheritance hierarchy
  – It may *not have any subclasses*
- Example:
  private final class Circle extends Shape

19 **Interfaces**

- Contains abstract method definitions needed by several classes and perhaps within

several class hierarchies
  – An alternate to declaring them in a superclass
- If a method is declared in an interface, all classes that "implement" the interface *must* declare a method with the same signature

20 **The Keyword interface**

- Used to *declare* an interface (replaces the keyword class in the header signature)
  – As with a class name, the name of the interface must be identical to the "*.java" filename
- Example:

```
public interface Color
{
    public abstract void setColor();
    public abstract String getColor();
}
```

  – Filename for the above must be "Color.java"

21 **Implementing Interfaces**

- Interfaces are *not inherited* in subclasses but rather they are *implemented*
- Classes may implement *several* interfaces …
  – Sort of like *multiple* inheritance …
  – Unlike subclasses which may inherit (extend) from *only one* superclass

22 **The Keyword implements**

- Used to implement an interface
- Format:

  public class *SubClassName* extends *SuperClassName* <u>implements</u> *InterfaceName1*[, *InterfaceName2*, …]
  { …
- Example:

```
public class Circle extends Shape implements Color
{ …
```

23 **Declaring Constants in Interfaces (Page 1)**

- Besides abstract method references, the only other elements that may be declared in interfaces are *constants*
- These constants can be accessed by *all classes* in which the interface is implemented
- The constant identifier must be:
  – Declared as final and may additionally be declared as static (they are static by default)
  – *Assigned a value* which may not be updated

24 **Declaring Constants in Interfaces (Page 2)**

- Format:

```
[public] [static] [final] type CONSTANT_NAME = value;
```

- Example:

```
public interface Color
{
    public static final String RED = "Red";
    public static final String LIGHT_BLUE = "Light Blue";
}
```

25 **Interface Programming Practice (Page 1)**

- According to the "Java Language Specification", in standard practice within an interface:
  - Methods are declared without the keywords public and abstract because these specifications are redundant
  - Constants are declared without the keywords public, static and final because they also are redundant

26 **Interface Programming Practice (Page 2)**

- Example:

```
public interface Color
{
    void setColor();
    String getColor();

    String RED = "Red";
    String LIGHT_BLUE = "Light Blue";
}
```