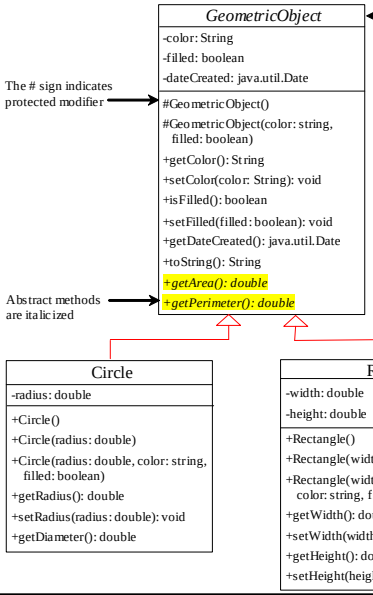
We are on the GUI fast track path...

- Chapter 13: Exception Handling – Skip for now
- Chapter 14: Abstract Classes and Interfaces
  - Sections 1 – 9: ActionListener interface
- Chapter 15: Graphics – Skip for now
- Chapter 16: Event-Driven Programming
  - Sections 1 – 7: Events, registering listeners, JTextField input
- Chapter 17: Creating Graphical User Interfaces
  - Sections 1 – 8: JButton, JCheckBox, JRadioButton, JLabel JTextField, JTextArea, JComboBox
- Chapter 18: Applets and Multimedia
  - Sections 1 – 7, 10, 11: Applets, Apps, and Audio
- Chapter 33: Swing Containers, Layouts, and Borders
  - Sections 1 – 3, 8: CardLayout, BoxLayout, null Layout, Border
- Chapter 34: Menus, Toolbars, and Dialogs

Ch 14: Abstract Classes and Interfaces

- Write Event Handler code to respond to user actions such as clicking a button
- Abstract Classes are general and not concrete
  - Cannot create new objects from Abstract Classes
  - Example: `public abstract class GeometricObject`
  - Abstract methods describe common methods for subclasses that will be defined in the subclass
    - Abstract methods make the class abstract
    - Denoted by Italics names in UML
    - Abstract methods are defined without implementation
    - `public abstract double getArea();`
  - Constructors of abstract class is defined protected
    - `protected GeometicObject();`
    - It may only be used by subclasses and objects cannot be created
- Polymorphic way to utilize commonly named methods

Abstract Classes and Abstract Methods



In a nonabstract subclass extended from an abstract class, all the abstract methods must be implemented, even if they are not used in the subclass.

A subclass can be abstract even if its superclass is concrete.

You cannot create an instance from an abstract class using the new operator, but an abstract class can be used as a data type.

 3

Interface in Java

- An interface is a class like program construct
  - Contains only constants and abstract methods
  - Similar to an abstract class, but the intent different
  - Interface specifies behavior for objects
  - For example, using interfaces objects specified as
    - Comparable
    - Edible
    - Cloneable

```
public interface InterfaceName
{
  constant declarations;
  method signatures;
}
```

  - interface is compiled into a separate bytecode file
  - Like an abstract class, you cannot create an instance from an interface using the new operator
  - An interface can be used as a data type for a variable and for casting

 4

Interface is a Special Class

```
1.  public interface Edible {
2.    /** Describe how to eat */
3.    public abstract String howToEat();
4.  }
```

```
1.  public class TestEdible {
2.    public static void main(String[] args) {
3.      Object[] objects = {new Tiger(), new Chicken(), new Apple()};
4.      for (int i = 0; i < objects.length; i++)
5.        if (objects[i] instanceof Edible)
6.          System.out.println(((Edible)objects[i]).howToEat());
7.    }
8.  }
9.  class Animal {
10.   // Data fields, constructors, and methods omitted here
11. }
12. class Chicken extends Animal implements Edible {
13.   public String howToEat() {
14.     return "Chicken: Fry it";
15.   }
16. }
17. class Tiger extends Animal {
18. }
19. abstract class Fruit implements Edible {
20.   // Data fields, constructors, and methods omitted here
21. }
22. class Apple extends Fruit {
23.   public String howToEat() {
24.     return "Apple: Make apple cider";
25.   }
26. }
27. class Orange extends Fruit {
28.   public String howToEat() {
29.     return "Orange: Make orange juice";
30.   }
31. }
```

You can now use the Edible interface to specify whether an object is edible.

This is accomplished by letting the class for the object implement this interface using the implements keyword.

For example, the classes Chicken and Fruit implement the Edible interface.

Omitting Modifiers in Interfaces

All data fields are *public final static* and all methods are *public abstract* in an interface.

For this reason, these modifiers can be omitted, as shown below:

```
public interface T1 {
  public static final int K = 1;

  public abstract void p();
}
```

Equivalent

```
public interface T1 {
  int K = 1;

  void p();
}
```

A constant defined in an interface can be accessed using syntax InterfaceName.CONSTANT_NAME (e.g., T1.K).

Copyright © 2012 R.M. Laurie 6

The Comparable Interface

- For comparing two objects of the same type
  - Java provides the comparable Interface

```
// This interface is defined in java.lang package
package java.lang;
public interface Comparable {
  public int compareTo(Object o);
}
```

- Many classes in Java library implement Comparable
  - Define a natural order for the objects

```
public class String extends Object
    implements Comparable {
  // class body omitted
}
```

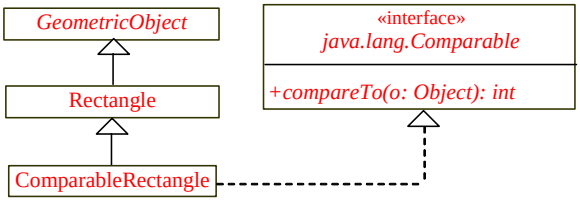
```
public class Date extends Object
    implements Comparable {
  // class body omitted
}
```

Many classes (e.g., String and Date) in the Java library implement Comparable to define a natural order for the objects. If you examine the source code of these classes, you will see the keyword implements used in the classes, as shown below:

 7

Defining Classes to Implement Comparable

Notation:
The interface name and the method names are italicized. The dashed lines and hollow triangles are used to point to the interface.




```
ComparableRectangle rectangle1 = new ComparableRectangle(4, 5);
ComparableRectangle rectangle2 = new ComparableRectangle(3, 6);
System.out.println(Max.max(rectangle1, rectangle2));
```

```
1.  public class ComparableRectangle extends Rectangle
2.      implements Comparable {
3.    /** Construct a ComparableRectangle with specified properties */
4.    public ComparableRectangle(double width, double height) {
5.      super(width, height);
6.    }
7.    /** Implement the compareTo method defined in Comparable */
8.    public int compareTo(Object o) {
9.      if (getArea() > ((ComparableRectangle)o).getArea())
10.       return 1;
11.     else if (getArea() < ((ComparableRectangle)o).getArea())
12.       return -1;
13.     else
14.       return 0;
15.   }
16. }
```

 8

ActionListener Interface for GUI

- Source object (e.g., button)
- Listener object contains method for processing the event, such as button click
  - Button is source object where action event occurs
  - Listener object handles the action event
    - Must be instance (object) of the ActionListener interface

```
1.  class OKListenerClass implements ActionListener {
2.    public void actionPerformed(ActionEvent e) {
3.      System.out.println("OK button clicked");
4.    }
5.  }
```

- ActionListener object must be registered with source object using method *source*.addActionListener(*listener*)

```
1.   public HandleEvent() {
2.      OKListenerClass listener1 = new OKListenerClass();
3.      jbtOK.addActionListener(listener1);
4.  }
```

 9

ActionListener HandleEvent

```
1.  import javax.swing.*;
2.  import java.awt.event.*;
3.  public class HandleEvent extends JFrame {
4.    public HandleEvent() {
5.      JButton jbtOK = new JButton("OK");
6.      JButton jbtCancel = new JButton("Cancel");
7.      JPanel panel = new JPanel();
8.      panel.add(jbtOK);
9.      panel.add(jbtCancel);
10.     add(panel); // Add panel to the frame

11.     // Register listeners
12.     OKListenerClass listener1 = new OKListenerClass();
13.     CancelListenerClass listener2 = new CancelListenerClass();
14.     jbtOK.addActionListener(listener1);
15.     jbtCancel.addActionListener(listener2);
16.   }
17.   public static void main(String[] args) {
18.     JFrame frame = new HandleEvent();
19.     frame.setTitle("Handle Event");
20.     frame.setSize(200, 150);
21.     frame.setLocation(200, 100);
22.     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.     frame.setVisible(true);
24.   }
25. }
26. class OKListenerClass implements ActionListener {
27.   public void actionPerformed(ActionEvent e) {
28.     System.out.println("OK button clicked");
29.   }
30. }
31. class CancelListenerClass implements ActionListener {
32.   public void actionPerformed(ActionEvent e) {
33.     System.out.println("Cancel button clicked");
34.   } }
```

2. Click OK

1. Start from the main method to create a window and display it

3. Click OK. The JVM invokes the listener's actionPerformed method

```
OK button clicked
Cancel button clicked
OK button clicked
```

The Cloneable Interfaces

- Marker Interface
  - An empty interface
  - marker interface does not contain constants or methods
  - It is used to denote that a class possesses certain desirable properties
  - A class that implements the Cloneable interface is marked cloneable, and its objects can be cloned using the clone() method from the Object class

```
package java.lang;
public interface Cloneable {
}
```

 11

Interfaces vs. Abstract Classes

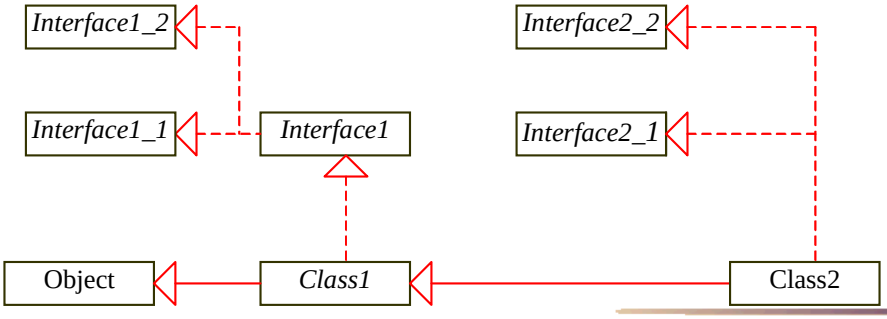
- Interface
  - data must be constants
  - methods have only signature and not implemented
  - Allows multiple extensions, unlike single inheritance of class
- Abstract class
  - data can have all types of data
  - abstract class can have concrete methods.

	Variables	Constructors	Methods
Abstract class	No restrictions	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.	No restrictions.
Interface	All variables must be public static final	No constructors. An interface cannot be instantiated using the new operator.	All methods must be public abstract instance methods

12

Interfaces vs. Abstract Classes

- All classes share a single root, the Object class
- Interfaces have no single root
- Like a class, an interface also defines a type. A variable of an interface type can reference any instance of the class that implements the interface.
- If a class extends an interface, this interface plays the same role as a superclass. You can use an interface as a data type and cast a variable of an interface type to its subclass, and vice versa.
- Suppose that c is an instance of Class2. c is also an instance of Object, Class1, Interface1, Interface1_1, Interface1_2, Interface2_1, and Interface2_2.



 13

Whether to use an interface or a class?

- **Abstract classes are best for:**
  - strong is-a relationship that clearly describes a parent-child relationship should be modeled using classes
  - For example, a staff member is a person. So their relationship should be modeled using class inheritance
- **Interfaces are best for:**
  - A weak is-a relationship, also known as an is-kind-of relationship, indicates that an object possesses a certain property
  - A weak is-a relationship can be modeled using interfaces.
  - You can also use interfaces to circumvent single inheritance restriction if multiple inheritance is desired.
  - In the case of multiple inheritance, you have to design one as a superclass, and others as interface.

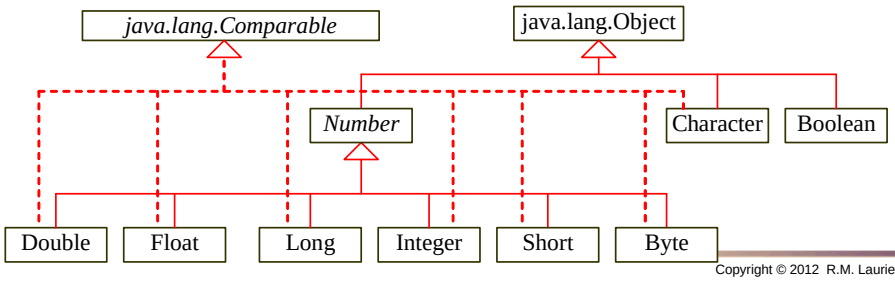
 14

Wrapper Classes

- Boolean
- Character
- Short
- Byte
- Integer
- Long
- Float
- Double

NOTES:

1. **The wrapper classes do not have no-arg constructors.**
2. **The instances of all wrapper classes are immutable, i.e., their internal values cannot be changed once the objects are created.**



 15