

CSE@UTA

Object-Oriented Java Programming

CSE3302 Programming Languages
Fall12007 ©Weimin He

Object-Oriented Java Programming

1

CSE@UTA

Why Java?

- Simple
- Platform Independent
- Safe
- Multi-Threaded
- Garbage Collected

CSE3302 Programming Languages
Fall12007 ©Weimin He

Object-Oriented Java Programming

2

CSE@UTA

Objects, Classes, and Methods

- **Object**
Something that occupies memory and has a state
- **Class**
Pattern (type) of object
- **Method**
Function and procedure to access object state

CSE3302 Programming Languages
Fall12007 ©Weimin He

Object-Oriented Java Programming

3

CSE@UTA

Constructors Overloading

```
public class Point { ... }
public class Rectangle {
    private int width = 0;
    private int height = 0;
    private Point origin;

    public Rectangle() {
        origin = new Point(0, 0);
    }

    public Rectangle(Point p, int w, int h) {
        origin = p;
        width = w;
        height = h;
    }
    ...
}
```

CSE3302 Programming Languages
Fall12007 ©Weimin He

Object-Oriented Java Programming

4

CSE@UTA

Access Control

- **Public:**
- **Private:**
- **Protected:**
- **Package:**

CSE3302 Programming Languages
Fall12007 ©Weimin He

Object-Oriented Java Programming

5

CSE@UTA

Example of Access Control

```
Base.java
package p1;
public class Base {
    private int u;
    protected int v;
    public int w;
    private void g () { System.out.println(v); }
    protected void f () { System.out.println(u); }
    g();
}

Derived.java
package p2;
import p1.Base;

public class Derived extends Base {
    void h () { f(); }
    void k () { g(); }
}

Test.java
package p2;
import p1.Base;

public class Test {
    public static void main(String[] args) {
        Base b = new Base();
        b.u = 1;
        b.v = 2;
        b.w = 3;

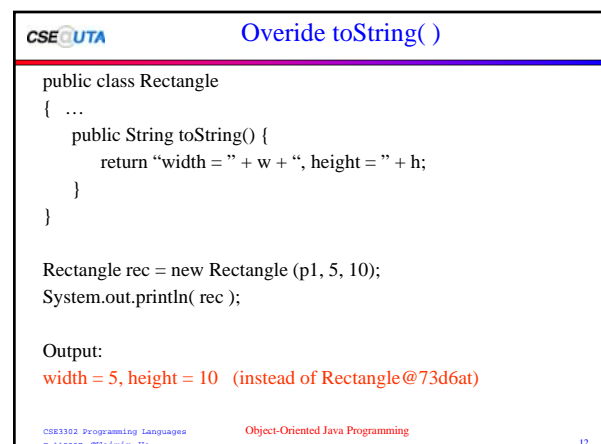
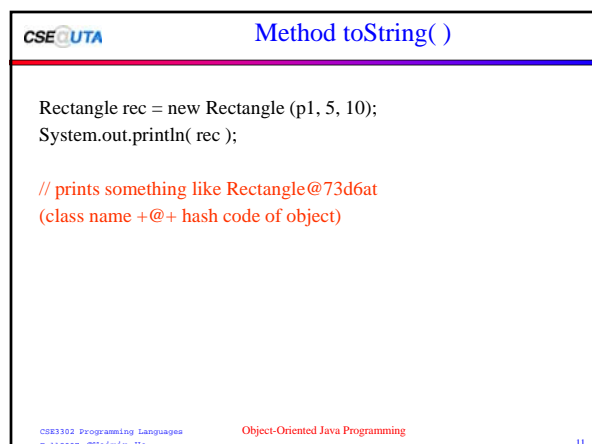
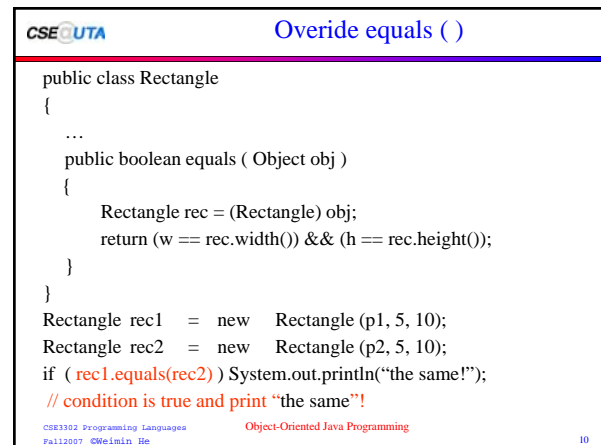
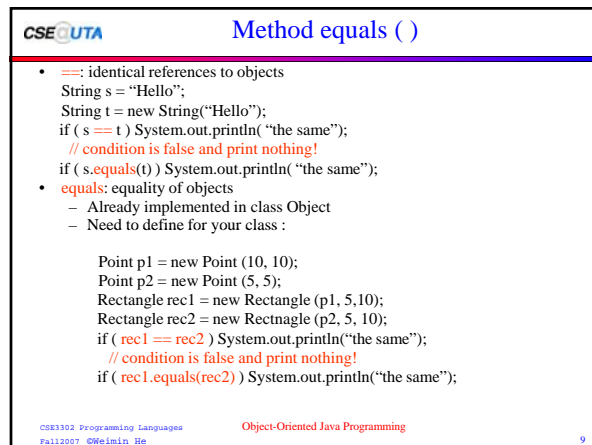
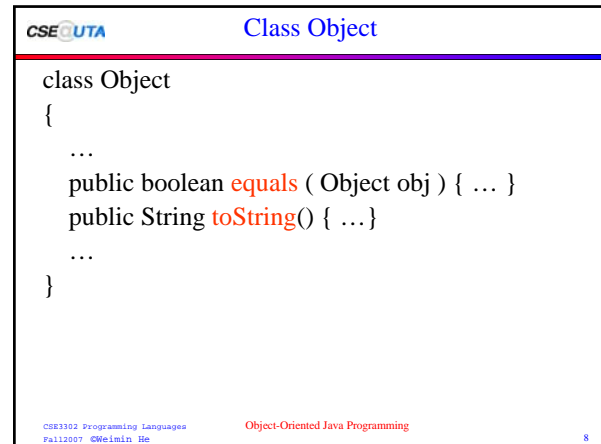
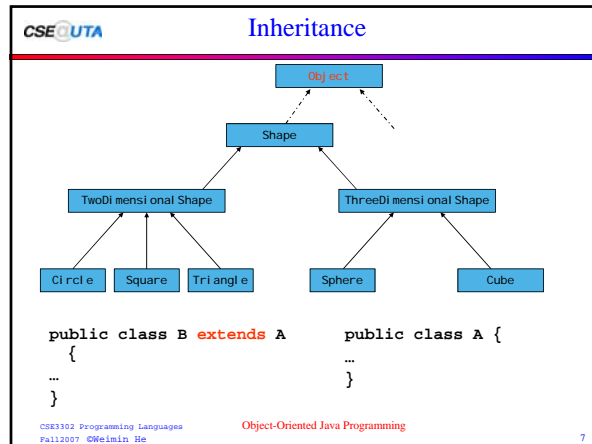
        b.g();
        b.f();
        Derived d = new Derived();

        d.h();
        d.k();
    }
}
```

CSE3302 Programming Languages
Fall12007 ©Weimin He

Object-Oriented Java Programming

6



CSE@UTA **Inheritance**

```

public class Circle
{ private Point center;
  private double radius;

  public Circle( Point c, double r)
  { center = c;
    radius = r;
  }

  public double area()
  { return Math.PI * radius* radius; }
}

public class Rectangle
{ private Point center;
  private double width;
  private double height;

  public Rectangle( Point c,
                   double w, double h)
  { center = c;
    width = w;
    height = h;
  }

  public double area()
  { return width * height; }
}

```

CSE3302 Programming Languages Fall12007 ©Weimin He **Object-Oriented Java Programming** 13

CSE@UTA **Inheritance (Cont'd)**

```

public abstract class Shape
{ private Point center;

  public Shape( Point c)
  { center = c; }

  public abstract double area();
}

```

CSE3302 Programming Languages Fall12007 ©Weimin He **Object-Oriented Java Programming** 14

CSE@UTA **Inheritance (Cont'd)**

```

public class Circle extends Shape
{ public Circle( Point c, double r)
  { super(c);
    radius = r;
  }

  public double area()
  { return Math.PI * radius* radius; }
}

public class Rectangle extends Shape
{ private double width;
  private double height;

  public Rectangle( Point c,
                  double w, double h)
  { super(c);
    width = w;
    height = h;
  }

  public double area()
  { return width * height; }
}

```

CSE3302 Programming Languages Fall12007 ©Weimin He **Object-Oriented Java Programming** 15

CSE@UTA **Casting**

- **Upcasting:**
Assign a subclass reference to a superclass variable
- **Downcasting:**
Assign Superclass Reference to subclass variable

CSE3302 Programming Languages Fall12007 ©Weimin He **Object-Oriented Java Programming** 16

CSE@UTA **Example of Casting**

```

public class Queue
{ // constructors and instance variables
  omitted
  public void enqueue( int x) { ... }
  public void dequeue() { ... }
  public int front() { ... }
  public boolean empty() { ... }
}

public class Deque extends Queue
{ ...
  public void addFront( int x) { ... }
  public void deleteRear() { ... }
}

Queue q1, q2;
Deque d;

d = new Deque();
q1 = d;
q1.deleteRear();
// Compilation error
d = q1;
// Compilation error
d = (Deque) q1;
d.deleteRear();
q2 = new Queue();
d = (Deque) q2;
// Runtime error
d.deleteRear();

```

CSE3302 Programming Languages Fall12007 ©Weimin He **Object-Oriented Java Programming** 17

CSE@UTA **Dynamic Binding**

- **Dynamic Binding**
 - A mechanism by which, when the compiler can't determine which method implementation to use in advance, the *runtime system* (JVM) selects the appropriate method at runtime, based on the class of the object.
 - The process of binding a call to a particular method. *This is performed dynamically at run-time.*

CSE3302 Programming Languages Fall12007 ©Weimin He **Object-Oriented Java Programming** 18

CSE@UTA **Dynamic Binding Example**

```

class A
{ void p() { System.out.println("A.p"); }
  void q() { System.out.println("A.q"); }
  void f() { p(); q(); }
}
class B extends A
{ void p() { System.out.println("B.p"); }
  void q() { System.out.println("B.q"); super.q(); }
}
public class Test
{ public static void main(String[] args)
  { A a = new A();
    a.f();
    a = new B();
    a.f(); }
}

```

Print:

A.p
A.q
B.p
B.q
A.q

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 19

CSE@UTA **Dynamic Binding Example (Cont'd)**

```

class A {
  public void q(){
    System.out.println("A.q()");
  }
}
class B extends A {
  public void f(){
    System.out.println("B.f()");
    super.q();
  }
}
class C extends B {
  public void q() {
    System.out.println("C.q()");
  }
}
public class D extends C {
  public static void main(String[] args){
    D d = new D();
    d.f();
  }
}

```

Print:

B.f()
A.q()

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 20

CSE@UTA **Abstract Class**

- Abstract classes
 - Cannot be instantiated
 - Incomplete: subclasses fill in "missing pieces"
- To make a class abstract
 - public **abstract** class Shape {...}
 - Contain one or more **abstract methods**
 - No implementation
 - E.g., public abstract void draw();
- Subclasses:
 - fill in "missing pieces" (i.e., overriding the abstract methods)
 - E.g., Circle, Triangle, Rectangle extends Shape
 - Each must implement draw it is concrete

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 21

CSE@UTA **Example of Abstract Class**

```

public class Base
{
  public String m1()
  { return "Base.m1"; }
}

public abstract class Derived extends Base
{
  public abstract String m2();
}

public class Derived2 extends Derived
{
  public String m2()
  { return "Derived2.m2"; }
}

```

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 22

CSE@UTA **Interface**

- No method implementations
- Java doesn't allow multiple inheritance:
 - E.g., ... C extends A, B ...
- Instead, use **Interface**
 - E.g., ... C **implements** I₁, I₂ ...
- One class may implement multiple interfaces
 - **Must implement all functions in those interfaces if class is concrete**

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 23

CSE@UTA **Interface Example**

```

public class Base
{
  public String m1()
  { return "Base.m1"; }
}

interface Interface1 { String m2(); }

interface Interface2 { String m3(); }

public class Derived extends Base implements Interface1, Interface2
{
  public String m2()
  { return "Derived.m2"; }

  public String m3()
  { return "Derived.m3"; }
}

```

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 24

CSE@UTA Interface Example (Cont'd)

```
interface Interface1 { String m( String s ); }

interface Interface2 { String m( int i ); }

public class Derived implements Interface1, Interface2 {
    public String m(String s) {
        return "Derived.Interface1.m";
    }
    public String m(int i) {
        return "Derived.Interface2.m";
    }
}
```

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 25

CSE@UTA Interface Example (Cont'd)

```
interface Interface1 { String m( String s ); }

interface Interface2 { String m( String s ); }

public class Derived implements Interface1, Interface2 {
    public String m(String s) {
        return "Derived.Interface1&Interface2.m";
    }
}
```

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 26

CSE@UTA Interface Example (Cont'd)

```
interface Interface1 { String m( String s ); }
interface Interface2 { void m( String s ); }

public class Derived implements Interface1, Interface2 {
    public String m(String s) {
        return "Derived.Interface1.m";
    }

    public void m(int i) {
        System.out.println("Derived.Interface2.m");
    }
}

Derived d = new Derived();
d.m(10); // Compilation error
```

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 27

CSE@UTA Interface (Java) vs. Multiple Inheritance (C++)

```
abstract class A {
    abstract public void f()
}
class B extends A {
    public void f(){
        System.out.println("B.f()");
    }
}
class C extends A {
    public void f() {
        System.out.println("C.f()");
    }
}
public class D extends B, C {
    public static void main(String[] args){
        D d = new D();
        d.f();
    }
}
```

In C++:

```
D* d = new D;
d->f();
// Compilation error

d->B::f();
// Legal
```

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 28

CSE@UTA Interfaces vs. Abstract Classes

- A class may implement several interfaces
- An interface cannot provide any code at all
- Static final constants only
- A class may extend only one abstract class
- An abstract class can provide partial code
- Both instance and static constants are possible

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 29

CSE@UTA Combination of Abstract Class and Interface

```
public class Base {
    public String m1() {
        return "Base.m1";
    }
}
interface Interface1 {
    String m1();
    String m2();
}
public abstract class Derived extends Base implements Interface1 {
    public String m2() {
        return "Derived.m2";
    }
}
public class Derived2 extends Derived {
    public String m1() {
        return "Derived2.m1";
    }
}
```

```
Derived2 derived2 = new Derived2();
Base base = derived2;

tmp = derived2.m1("Hello");
// tmp is "Derived2.m1"

tmp = base.m1("Hello");
// tmp is "Derived2.m1"

tmp = derived2.m2();
// tmp is "Derived.m2"
```

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 30

CSE@UTA String Processing

- Class `java.lang.String`
- Class `java.lang.StringBuffer`
- Class `java.util.StringTokenizer`
- URL:
 - <http://java.sun.com/j2se/1.4.2/docs/api/>

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 31

CSE@UTA Important Methods in Class String

- Method `length`
 - Determine `String` length
 - Like arrays, `Strings` always “know” their size
 - Unlike array, `Strings` do not have length instance variable
- Method `charAt`
 - Get character at specific location in `String`
- Method `getChars`
 - Get entire set of characters in `String`
- Method `startsWith`
 - Tests if this string starts with the specified prefix
- Method `split`
 - Splits this string around matches of the given regular expression

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 32

CSE@UTA Important Methods in Class String

- Comparing `String` objects
 - Method `equals`
 - Method `equalsIgnoreCase`
 - Method `compareTo`
- Search for characters in `String`
 - Method `indexOf`
 - Method `lastIndexOf`
- Create `Strings` from other `Strings`
 - Method `substring`
- Concatenate two `String` objects Method `concat`
 - Method `concat`
- `String` provides static class methods
 - Method `valueOf`
 - Returns `String` representation of object, data, etc.

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 33

CSE@UTA Class `StringBuffer`

- Class `StringBuffer`
 - When `String` object is created, its contents cannot change
 - Used for creating and manipulating dynamic string data
 - i.e., modifiable `Strings`
 - Can store characters based on capacity
 - Capacity expands dynamically to handle additional characters
 - Uses operators `+` and `+=` for `String` concatenation

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 34

CSE@UTA Important Methods in `StringBuffer`

- Method `charAt`
 - Return `StringBuffer` character at specified index
- Method `setCharAt`
 - Set `StringBuffer` character at specified index
- Method `getChars`
 - Return character array from `StringBuffer`
- Method `append`
 - Allow data values to be added to `StringBuffer`
- Method `reverse`
 - Reverse `StringBuffer` contents
- Method `insert`
 - Allow data-type values to be inserted into `StringBuffer`
- Methods `delete` and `deleteCharAt`
 - Allow characters to be removed from `StringBuffer`

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 35

CSE@UTA Important Methods in Class `StringBuffer`

- Method `length`
 - Return `StringBuffer` length
- Method `capacity`
 - Return `StringBuffer` capacity
- Method `setLength`
 - Increase or decrease `StringBuffer` length
- Method `ensureCapacity`
 - Set `StringBuffer` capacity
 - Guarantee that `StringBuffer` has minimum capacity

CSE3302 Programming Languages Fall12007 ©Weimin He Object-Oriented Java Programming 36

CSE@UTA **Class StringTokenizer**

- **Tokenizer**
 - Partition `String` into individual substrings
 - Use *delimiter*
 - Java offers `java.util.StringTokenizer`

CSE3302 Programming Languages Fall12007 ©Weimin He **Object-Oriented Java Programming** 37

CSE@UTA **Important Methods in Class StringTokenizer**

- Constructor **`StringTokenizer(String str)`**
 - Constructs a string tokenizer for the specified string
- Constructor **`StringTokenizer(String str, String delim)`**
 - Constructs a string tokenizer for the specified string
- Method **`hasMoreTokens()`**
 - Tests if there are more tokens available from this tokenizer's string
- Method **`nextToken()`**
 - Returns the next token from this string tokenizer

CSE3302 Programming Languages Fall12007 ©Weimin He **Object-Oriented Java Programming** 38