Polymorphism

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, May 03-04, 2010

Method Invocation Polymorphism

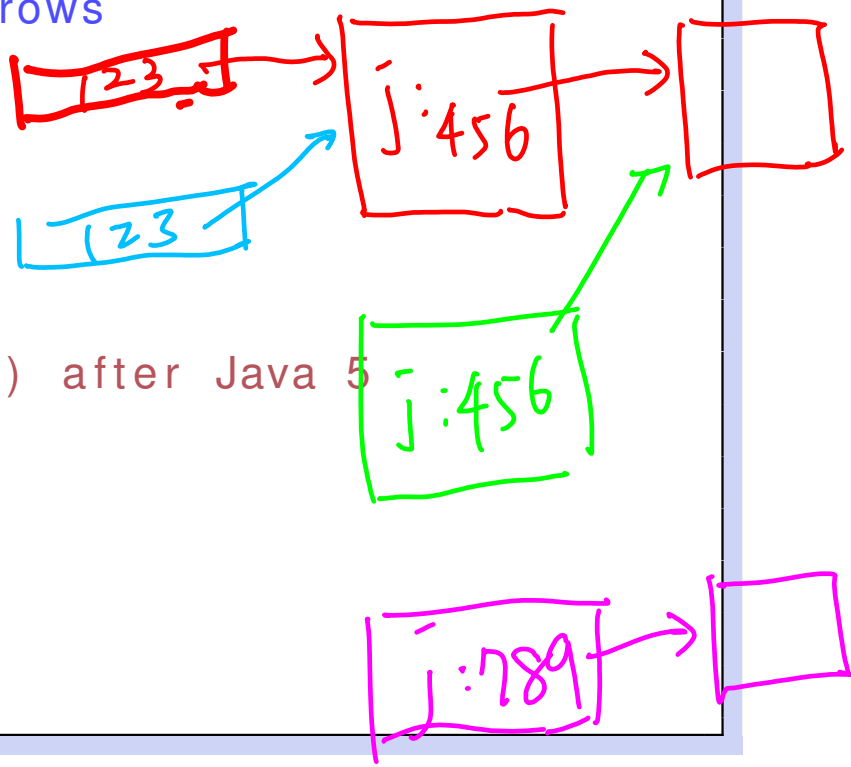
one method (via upper-level reference),
many possible extended behaviors

Object.equals

```
1  class Object{
2    public boolean equals(Object obj)
3    {
4      return this == obj;
5    }
6  }
7  class Coordinate extends Object{
8    double x, y;
9    public boolean equals(Object o){//must be public
10     if (o instanceof Coordinate){
11       Coordinate c = (Coordinate)o;
12       return (c.x == this.x && c.y == this.y);
13     }
14     else return false;
15   }
16 }
```

Object.clone

```
1   class Object{
2     protected native Object clone() throws
          CloneNotSupportedException;
3   }
4   class Professor extends Object{
5     private Joke j;
6
7     public Object clone(){
8     //possibly public Professor clone() after Java 5
9       Professor p = new Professor();
10      p.j = this.j;
11      //p.j = this.j.clone();
12      return p;
13    }
14  }
```



Object.toString

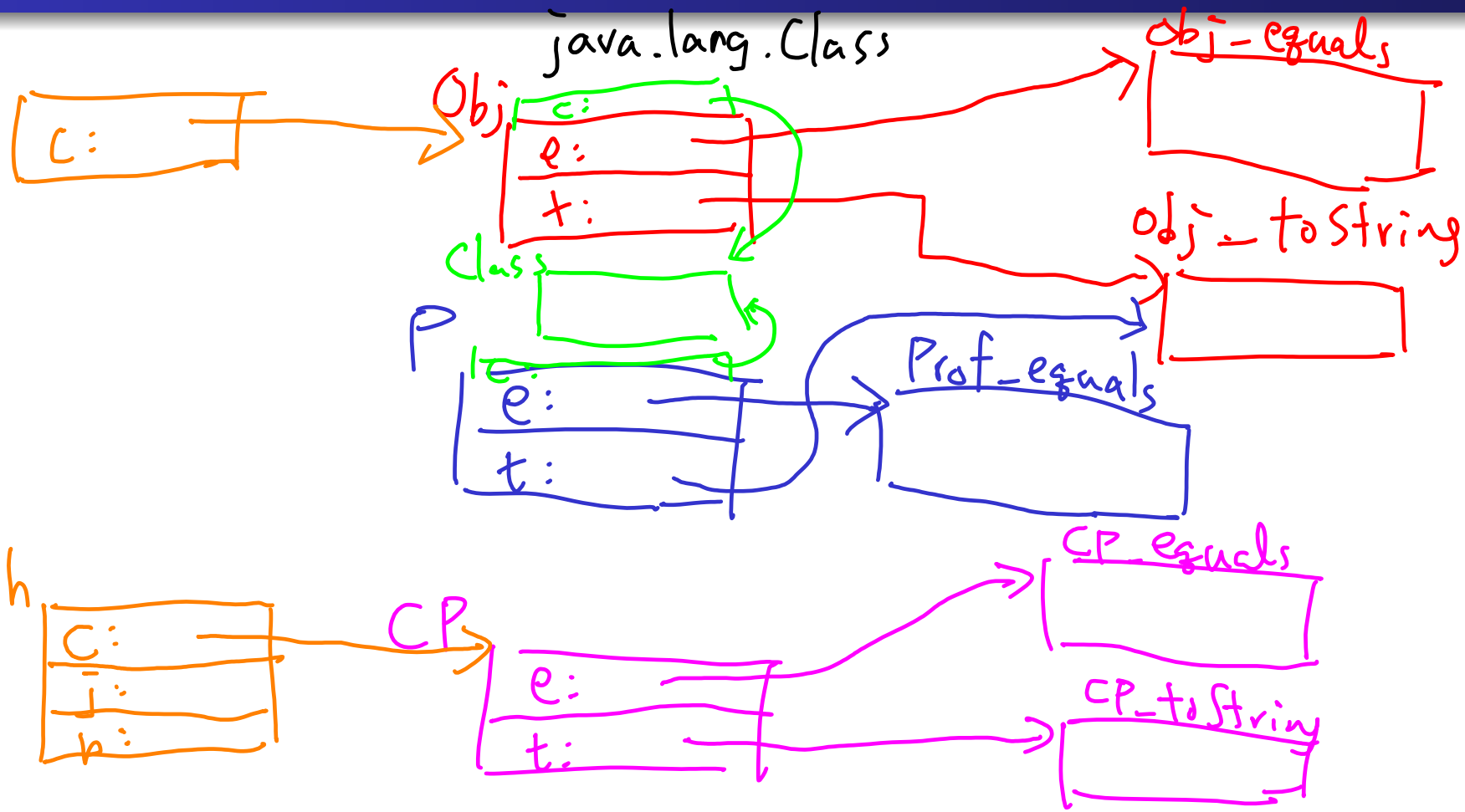
```
1  //usage
2  Coordinate c = new Coordinate();
3  System.out.println(c);
4  //from PrintStream.java
5  private synchronized void print (String str, boolean println)
6  { /* ... */ }
7  public void println (Object obj)
8  {
9    print(obj == null ? "null" : obj.toString(), true);
10 }
11 public void println (String str)
12 {
13   print (str == null ? "null" : str, true);
14 }
15 //BTW, does System.out.println("lalala")
16 //call PrintStream.println(Object)
17 //or PrintStream.println(String)?
18 //does System.out.println(null)
19 //call PrintStream.println(Object)
20 //or PrintStream.println(String)?
21 //does System.out.println((Object)null)
22 //call PrintStream.println(Object)
23 //or PrintStream.println(String)?
```

Twist Revisited

```
1  System.out.print("abc");
2  System.out.print(3);
3  System.out.print(4.0);
4  //a twist (of course, not exactly workable)
5  String("abc").print();
6  Integer(3).print();
7  Double(4.0).print();
```

System.out.print(Object) can have polymorphic behavior by internally calling the updated Object.toString() without overloading

V-Table: A Possible Mechanism of Method Overriding



again, not the only mechanism

RTTI revisited

- all we need is a link to the class area (stores name, vtable, etc.)
- where is the link? `java.lang.Object`
- how to access? `Object.getClass()`
- each area is an instance of `java.lang.Class`

Summary on Polymorphism

- one thing, many shapes
- important in strongly-typed platforms with inheritance
- view from content: one advanced content with many compatible access
- view from reference: one compatible reference can point to many advanced contents
- view from method: one compatible method “contract”, many different method “realization”

Abstract Class (1/3)

```
1  public class Professor(){
2    public void teach(){
3      System.out.println("not␣sure␣of␣what␣to␣teach!");
4    }
5  }
6  class CSIEProfessor extends Professor{
7    private void teach_oop(){ /* lalala */ }
8    public void teach(){ teach_oop(); }
9  }
10 class EEProfessor extends Professor{
11   private void teach_elec(){ /* lululu */ }
12   public void teach(){ teach_elec(); }
13 }
14 //in other places
15 Professor p = new Professor();
16 p.teach(); //not sure of what to teach!
```

- `teach` is a place-holder in Professor, expected to be overridden
- allows constructing a professor without any teaching ability! —absurd in some sense

Abstract Class (2/3)

```
1  public abstract class Professor{
2    public abstract void teach();
3  }
4  class CSIEProfessor extends Professor{
5    private void teach_oop(){ /* lalala */ }
6    public void teach(){ teach_oop(); }
7  }
8  class EEProfessor extends Professor{
9    private void teach_elec(){ /* lululu */ }
10   public void teach(){ teach_elec(); }
11 }
12 //in other places
13 Professor p = new Professor(); //hahaha!!
14 Professor p = new CSIEProfessor(); //okay
```

- `teach` is a place-holder in Professor, expected to be overridden
- but cannot construct a pure Professor instance anymore!

Abstract Class (3/3)

- abstract method [method not implemented]
  $\Rightarrow$ abstract class [cannot construct instance]?
- abstract class $\Rightarrow$ abstract method?
- public abstract method?
- proteced abstract method?
- private abstract method?
- keep being an abstract method in the subclass?
- concrete method(s) in an abstract class?
- instance variable(s) in an abstract class?
- static field(s) in an abstract class?
- constructor(s) in an abstract class?
- reference to an abstract class?

Key Point: Abstract Class

a **contract** for future extensions

Final Words

static {

}

- `static final` variable: accessed through class, and assigned once (in declaration or static constructor)
- `final` instance variable: accessed through instance, and assigned once (in declaration or every instance constructor)
- `final` instance method: cannot be overriden ($\approx$ assigned once)
- `static final` method: cannot be hidden by inheritance ($\approx$ assigned once)
- `final` class: cannot be inherited (and hence all methods final)