Polymorphism

Hsuan-Tien Lin

Deptartment of CSIE, NTU

OOP Class, May 5, 2009

Polymorphism: The Motto

水：

錢：coin / bill

球：baseball / basketball / ...

故事：小說 / 電影

One Thing, Many Shapes.....

brainstorm: where do we encounter this motto in the real world?

燈：水銀 / 鎢絲

人：角色

項鍊

One Variable, Many Values

```
1  char a;
2  switch(a){
3  case 1:
4    return 1 * 1;
5  case 2:
6    return 2 * 2;
7  case 3:
8    return 3 * 3;
9  ...
10 case 127:
11   return 127 * 127;
12 }
```

return a*a;

- better ways?
- mechanism? raw memory interpretation

One Class, Many Instances

class Student {
int score;
}

show(s.score);

```
1  Student s;
2  if (s equals student1)
3    show(score1);
4  else if (s equals student2)
5    show(score2);
6  ...
7  else if (s equals student100)
8    show(score100);
```

- better ways?
- mechanism? abstraction

One Method Name, Many Parameter Lists

```
1        //no overloading
2        System.out.printString("abc");
3        System.out.printInt(3);
4        System.out.printDouble(4.0);
5        //overloading
6        System.out.print("abc");
7        System.out.print(3);
8        System.out.print(4.0);
```

- mechanism? signature by name + parameter types

A Twist in Method Overloading

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

- mechanism? just write print() for every class (we'll see)

Polymorphic Behavior

- all cases above: some polymorphic behavior
- **BUT** almost no one calls them real “polymorphism”

Why Not?

Polymorphic Behavior on Known Stuff

- polymorphic behavior of **known** primitive-type variables
- polymorphic behavior of **known** classes (extended types)
- polymorphic behavior of **known** parameter types

Unknown Stuff: Future Extensions

Backward Compatibility

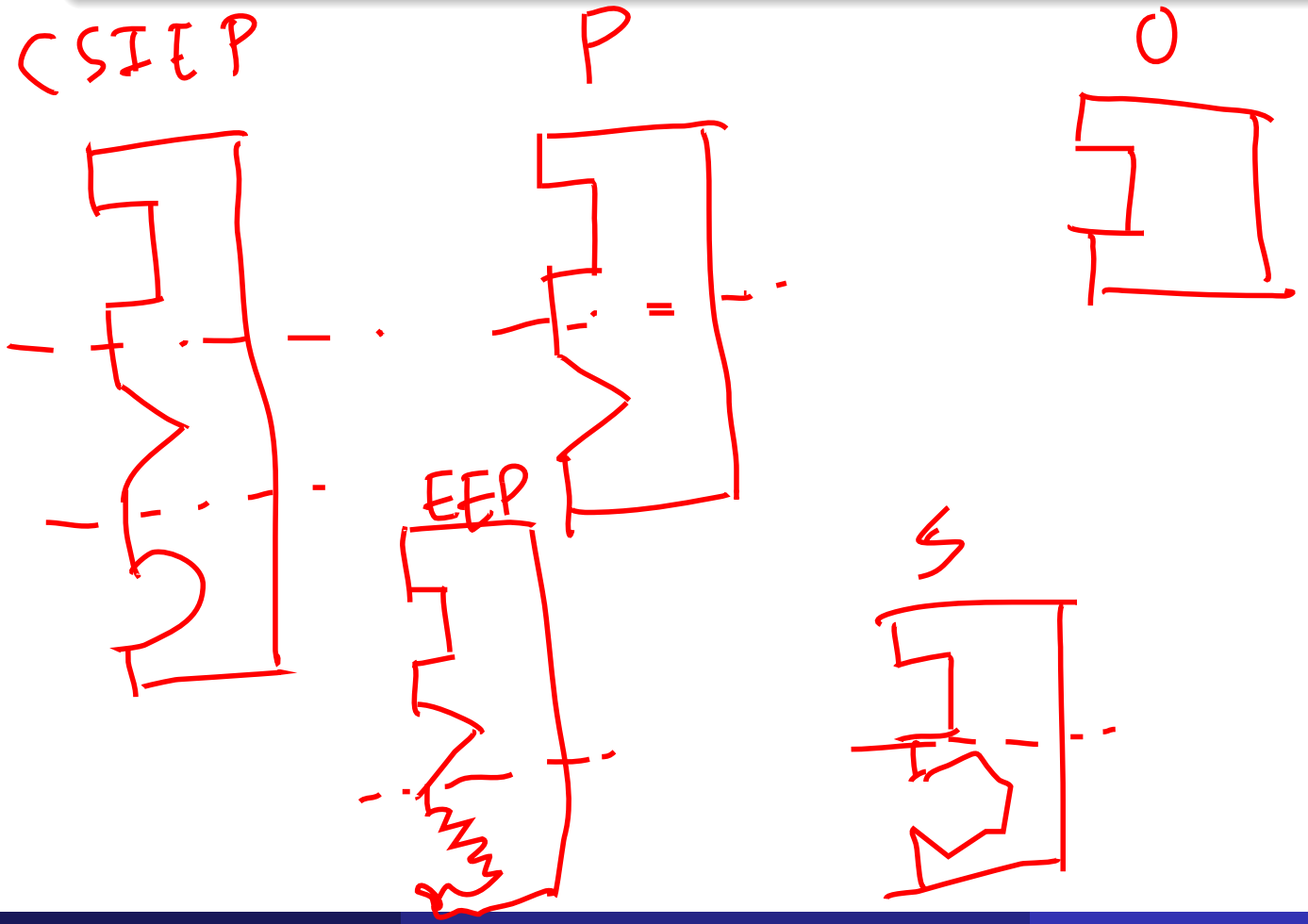
inheritance hierarchy

Forward Advance

newly added variables/methods

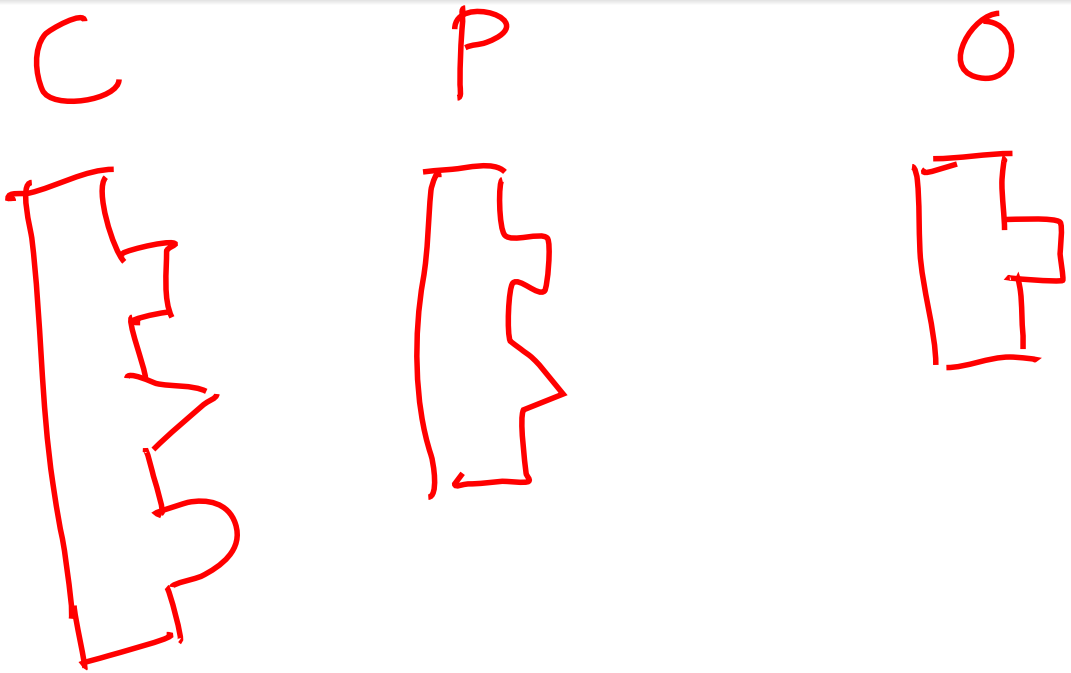
Polymorphism of Object Content

one **advanced** content, many **compatible** ways to access



Polymorphism of Object References

one **compatible** reference, many **advanced** contents to point to



Reference Upcast versus Reference Downcast

Upcast

simple (backward compatibility)

Downcast

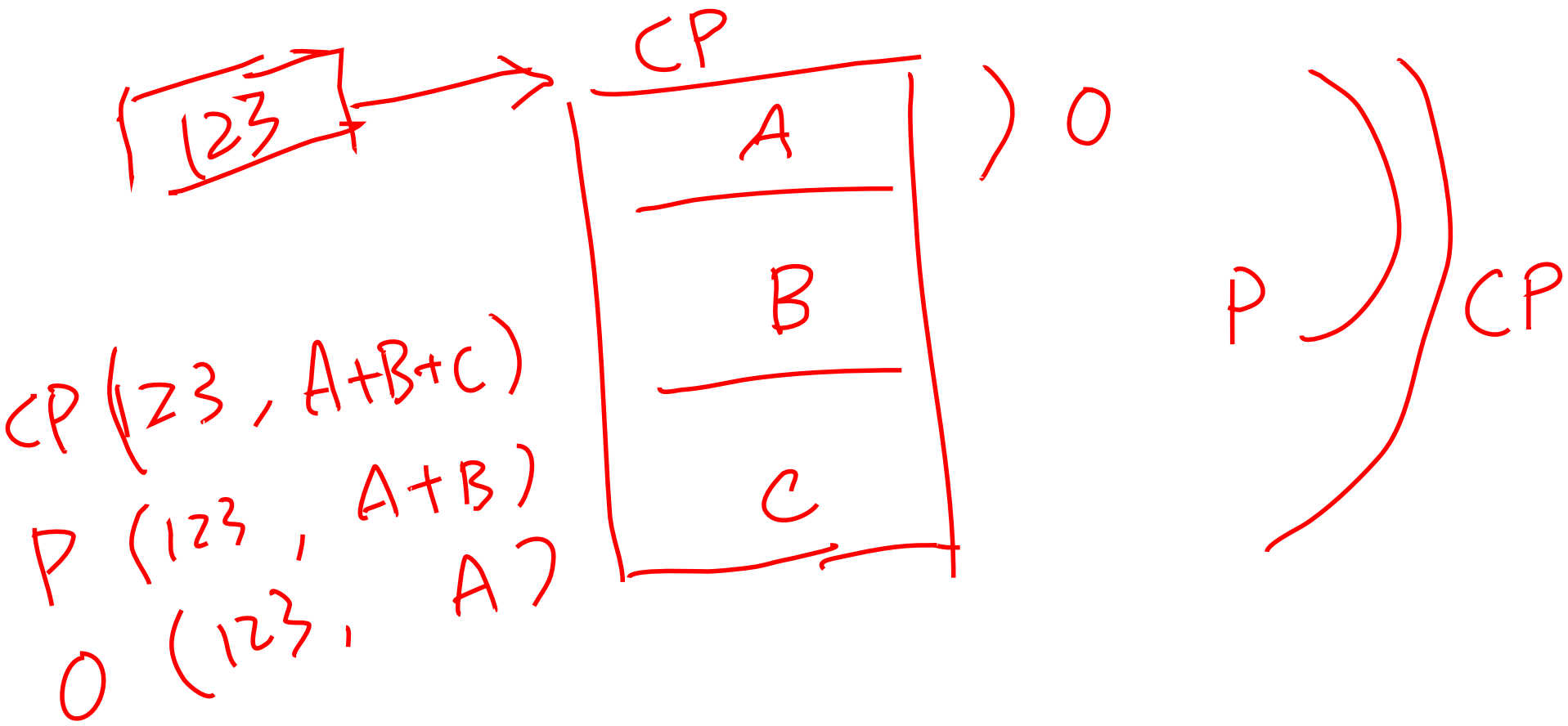
need to check whether content fits (forward advance)

need RTTI (run-time type information/identification) to make downcast work **(where did we see it?)**

instanceof

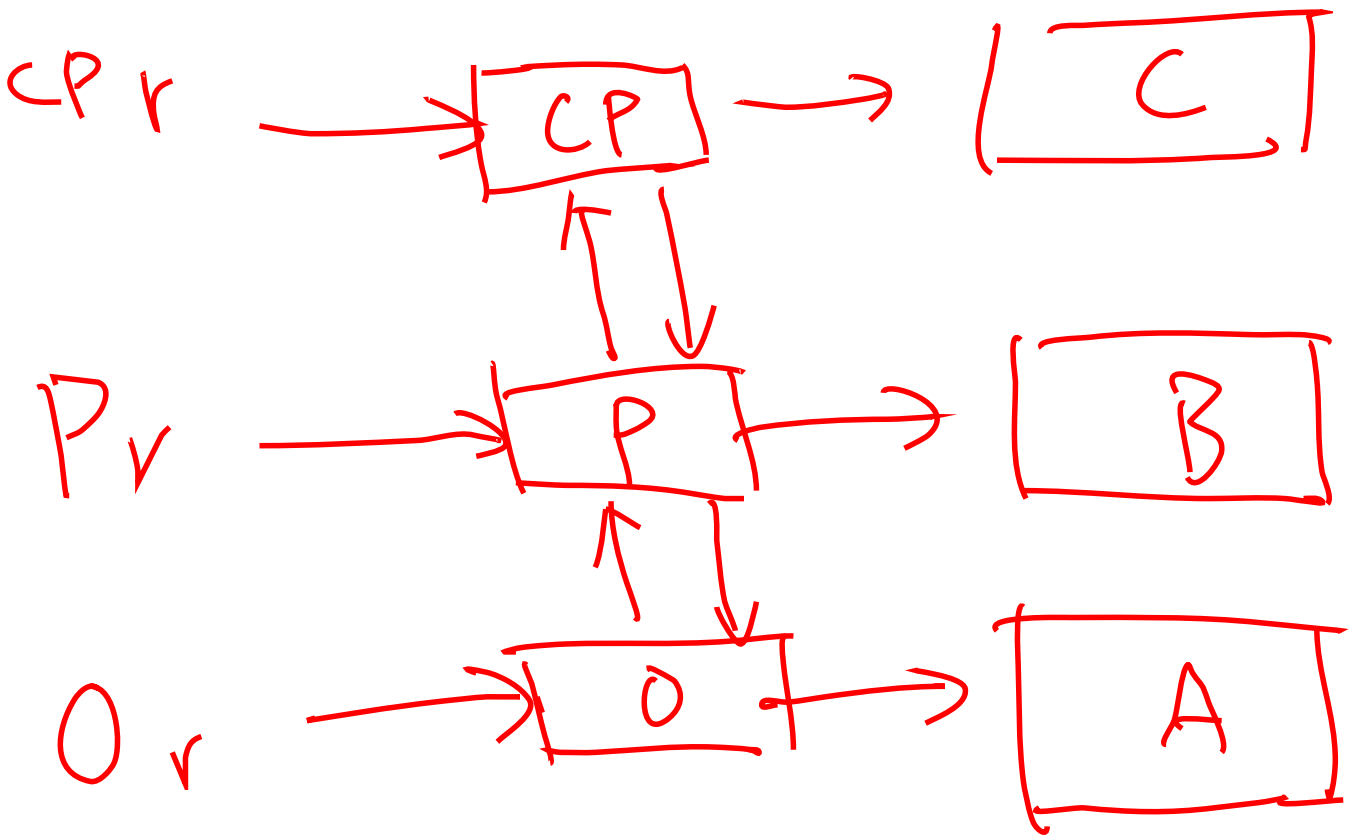
Content/Reference Polymorphism

a possible mechanism: shared prefix



Content/Reference Polymorphism

but not the only mechanism: how about chains?



Content/Reference Polymorphism: Summary

backward compatibility handled

forward advance: only via downcast (RTTI)

simpler mechanism for forward advance?

Our Needs in Forward Advance

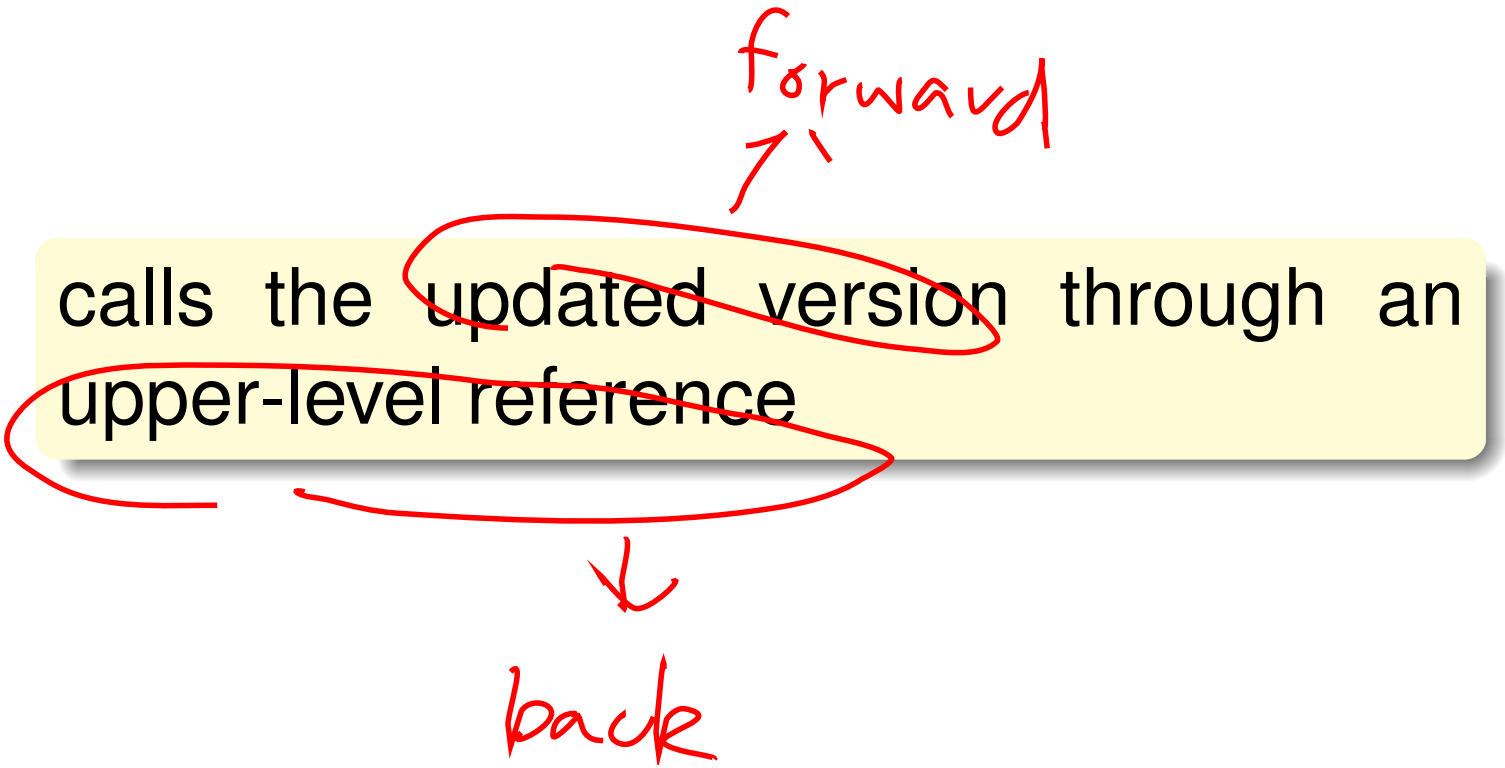
- 增加新的變數
- 增加新的方法
- 改變原有的方法
- 改變變數的值
- 增加新 class

砍掉 / 重練 → 更好 design / 更好工具

Our Needs in Forward Advance

- new instance variables for advanced state
- new instance methods to manipulate new variables
- give new meanings to existing instance variables
- give new meanings to existing instance methods
- write an "updated but compatible" version of existing method

Method Overriding (Virtual Function)



Method Invocation Polymorphism

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

Object.equals

Coordinate c1 ; new .x .y
Coordinate c2 ;
c1.equals(c2)
Object o = c1;
o.equals(c2);

```
1  class Coordinate extends Object{
2    double x, y;
3
4    bool equals(Object o){
5      if (o instanceof Coordinate){
6        Coordinate c = (Coordinate)o;
7        return (c.x == this.x && c.y == this.y);
8      }
9      return false;
10   }
11 }
```

boolean

Object.toString

override toString

```
1        Coordinate c = new Coordinate();
2        System.out.println(c);
```

c is Coordinate-type

⇒ c is Object-type

⇒ print_Obj invoked

⇒ Object.toString() ←

⇒ print_Str invoked

Twist Revisited

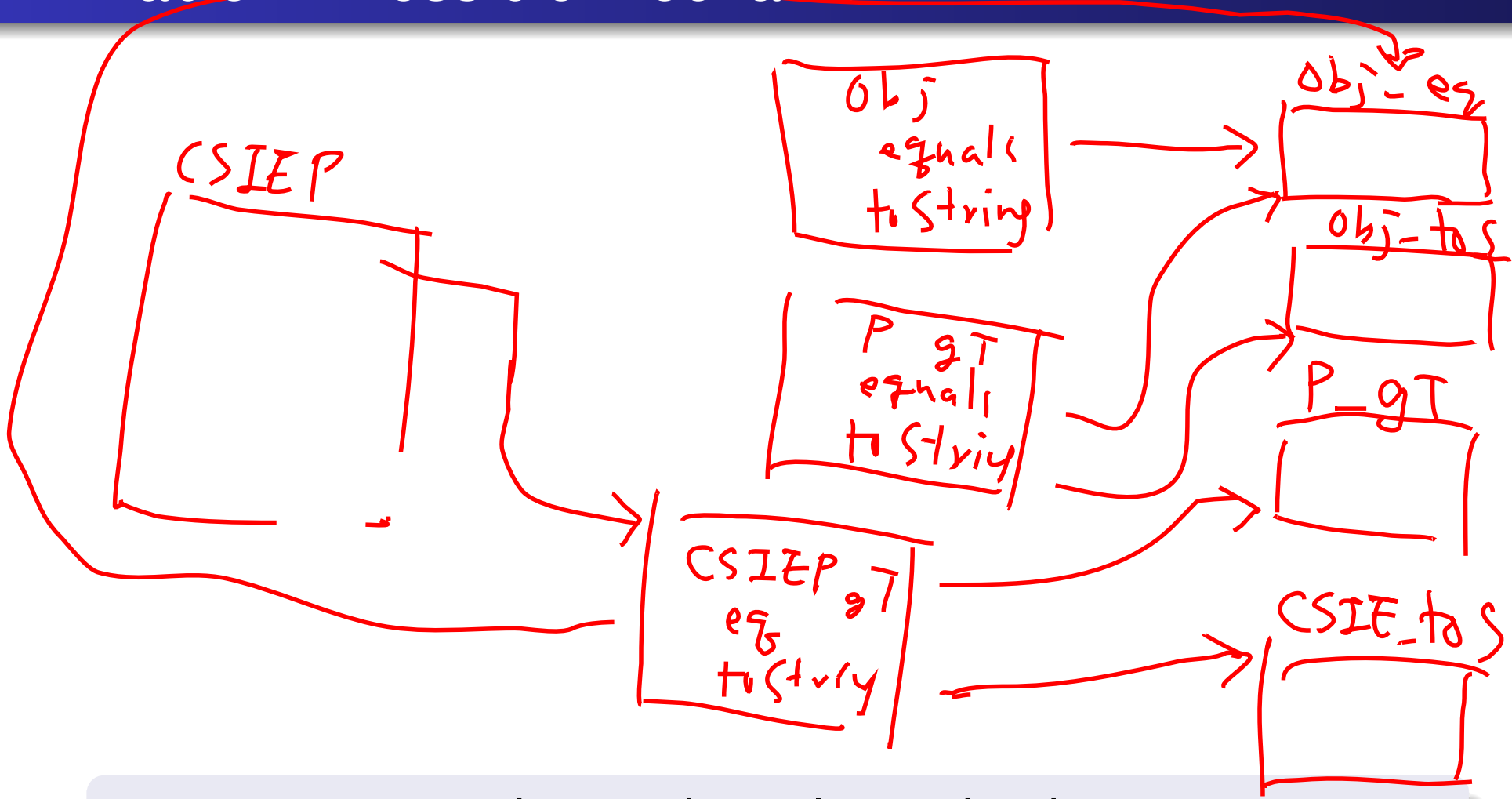
```
1       System.out.print("abc");
2       System.out.print(3);
3       System.out.print(4.0);
4       //a twist (of course, not exactly workable
5       String("abc").print();
6       Integer(3).print();
7       Double(4.0).print();
```

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

A Trip with `java.lang.Object`

A Trip with `java.lang.String`

V-Table: A Possible Mechanism



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`

An Advanced Use

How about creating an object with dynamic class name?!

see HW5

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”