CSc 133 Lecture Notes

V - Polymorphism

Computer Science Department
California State University, Sacramento

Overview

- Definitions
- Static ("compile-time") Polymorphism
- Polymorphic references, Upcasting / Downcasting
- Runtime ("dynamic") Polymorphism
- Polymorphic Safety
- Polymorphism - Java vs. C++

Polymorphism Defined

- Literally: from the Greek

  poly *("many")* + ***morphos*** *("forms")*

- Examples in nature:
  - Carbon: graphite or diamond
  - H_2O: water, ice, or steam
  - Honeybees: queen, drone, or worker

- Programming examples:
  - An operation that can be done on various types of objects
  - An operation that can be done in a variety of ways
  - A reference can be assigned to different types

"Static" Polymorphism

Detectable *during compilation.*

Example: Operator overloading:

```
int1 =  int2 + int3 ;

float1 =  float2 + float3 ;
```

- The "+" can perform on *different* types of objects
- "+" can therefore be thought of as a "*polymorphic operator*"

"Static" Polymorphism (cont.)

Another example: Method overloading:

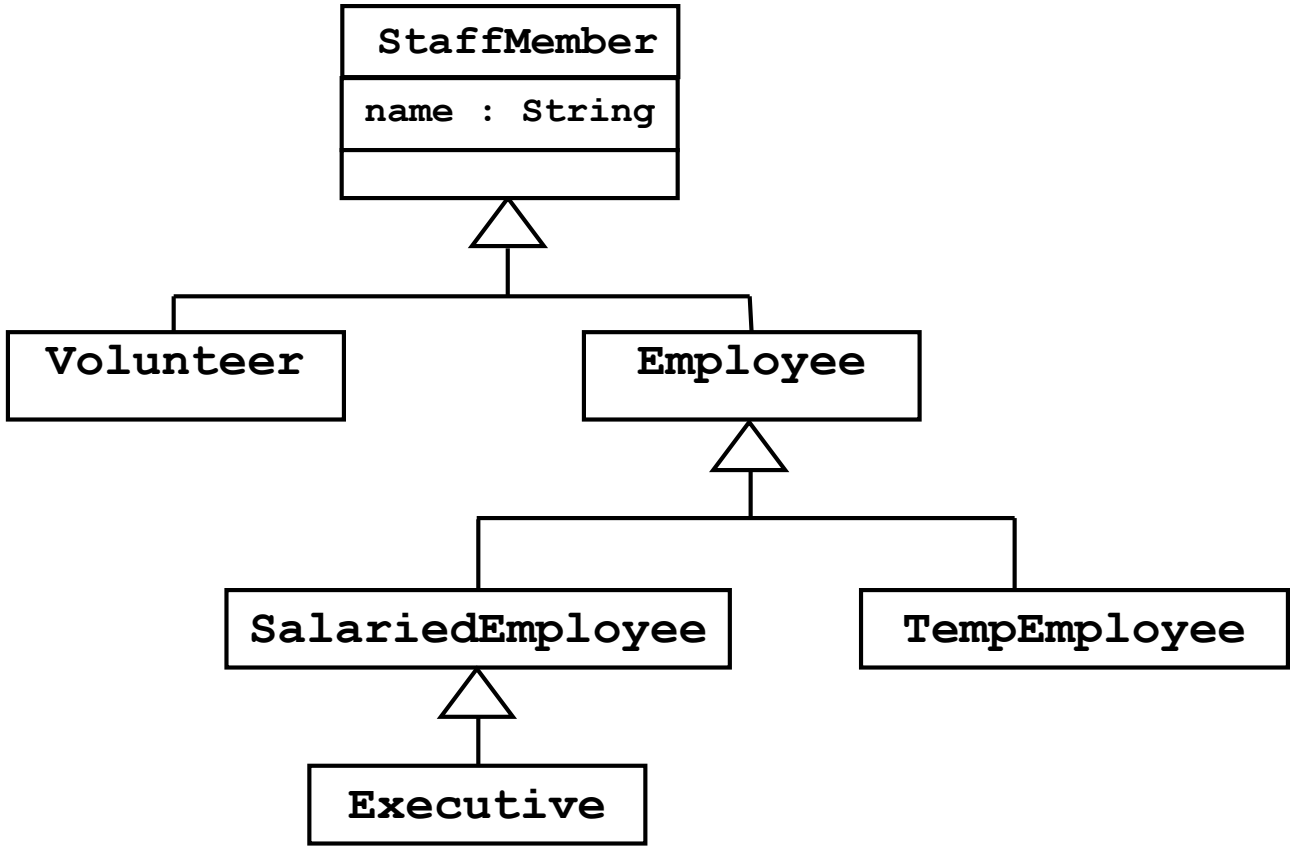
```
//return the distance to an origin
double distance (int x, int y) { . . . }

//return the distance between two points
double distance (Point p1, Point p2) { . . . }
```

- Same method name, for two different operations
- "**`distance`**" can therefore be thought of as a "*polymorphic method*"

Polymorphic References

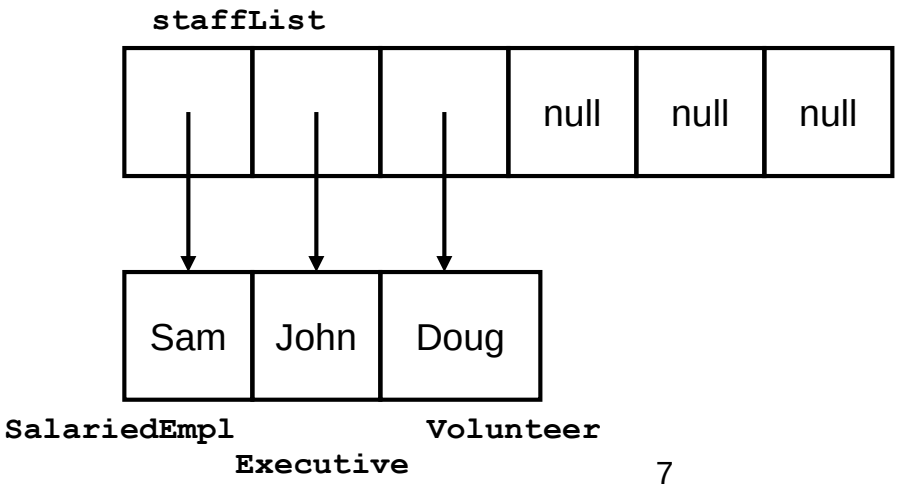
Consider the following class hierarchy:



Polymorphic References (cont.)

- A "polymorphic reference" can *refer to different object types at runtime*:

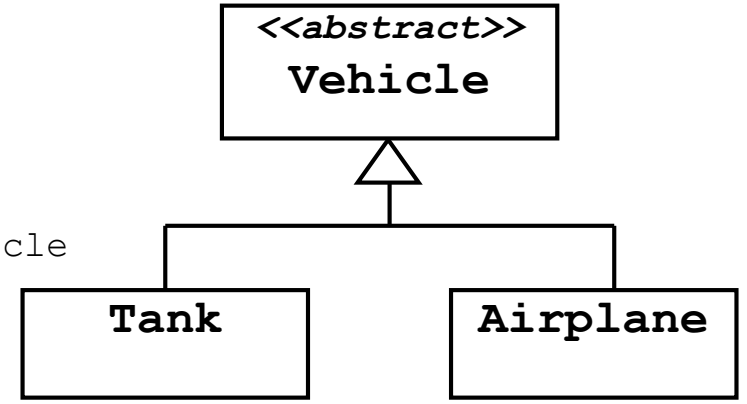
```
StaffMember [ ] staffList = new StaffMember[6];
. . .
staffList[0] = new SalariedEmployee ("Sam");
staffList[1] = new Executive ("John");
staffList[2] = new Volunteer ("Doug");
. . .
```



Upcasting and Downcasting

- "Upcasting" allowed in assignments:

```
Vehicle v ;
Airplane a = new Airplane();
Tank t = new Tank();
...
v = t ;   // a tank IS-A Vehicle
v = a;    // an airplane IS-A Vehicle
```

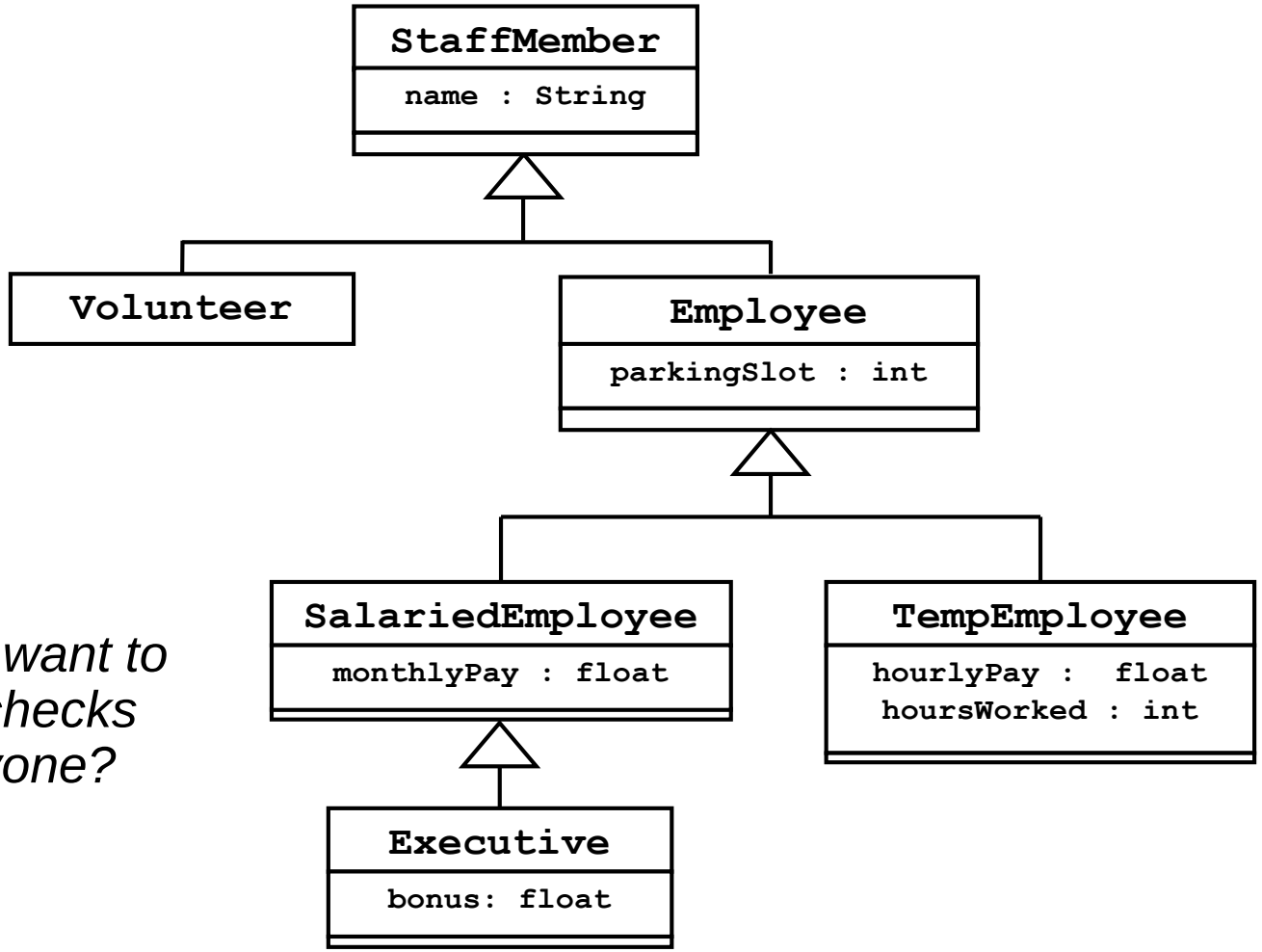


- "Downcasting" requires casting:

```
t = v ;          // compiler error - a Vehicle isn't a Tank
t = (Tank) v ;   // legal, but dangerous
```

Runtime Polymorphism

Consider this expanded version of the hierarchy shown earlier:



What if we want to print paychecks for everyone?

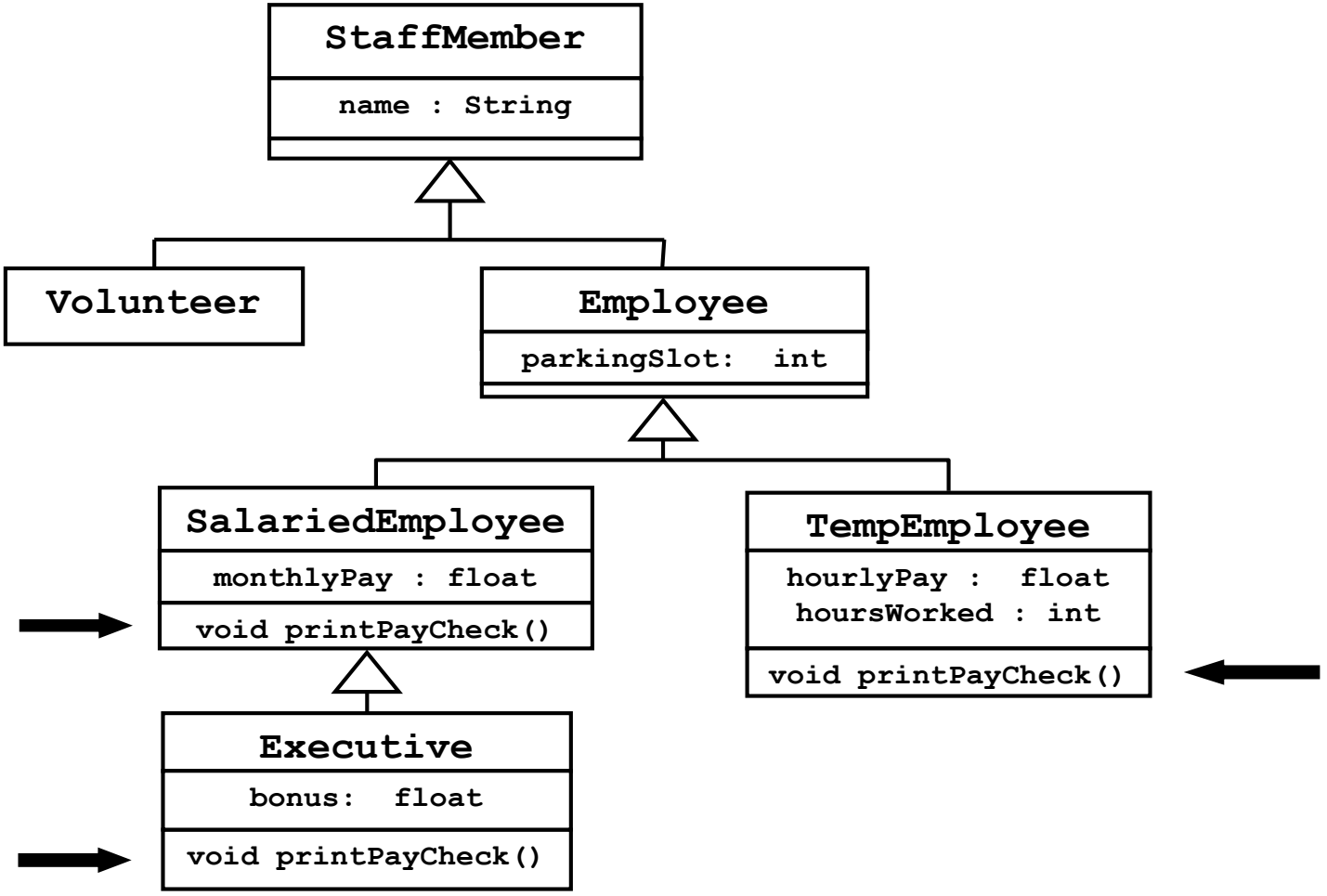
Runtime Polymorphism (cont.)

Printing Paychecks (traditional approach) :

```
for (int i=0; i<staffList.length; i++) {
  String name = staffList[i].getName();
  float amount = 0;
  if (staffList[i] instanceof SalariedEmployee) {
      SalariedEmployee curEmp = (SalariedEmployee) staffList[i];
      amount = curEmp.getMonthlyPay();
      printPayCheck (name, amount);
  } else if (staffList[i] instanceof Executive) {
      Executive curExec = (Executive) staffList[i] ;
      amount = curExec.getMonthlyPay() + curExec.getBonus());
      printPayCheck (name, amount);
  } else if (staffList[i] instanceof TempEmployee) {
      TempEmployee curTemp = (TempEmployee) staffList[i] ;
      amount = curTemp.getHoursWorked()*curTemp.getHourlyPay();
      printPayCheck (name, amount);
  }
}
. . .
private void printPayCheck (String name, float amt) {
  System.out.println ("Pay To The Order Of:" + name + " $" + amt);
}
```

Runtime Polymorphism (cont.)

First, paycheck computation should be "encapsulated":



Runtime Polymorphism (cont.)

Polymorphic solution:

```
...
for (int i=0; i<staffList.length; i++) {
     staffList[i].printPayCheck() ;
}
...
```

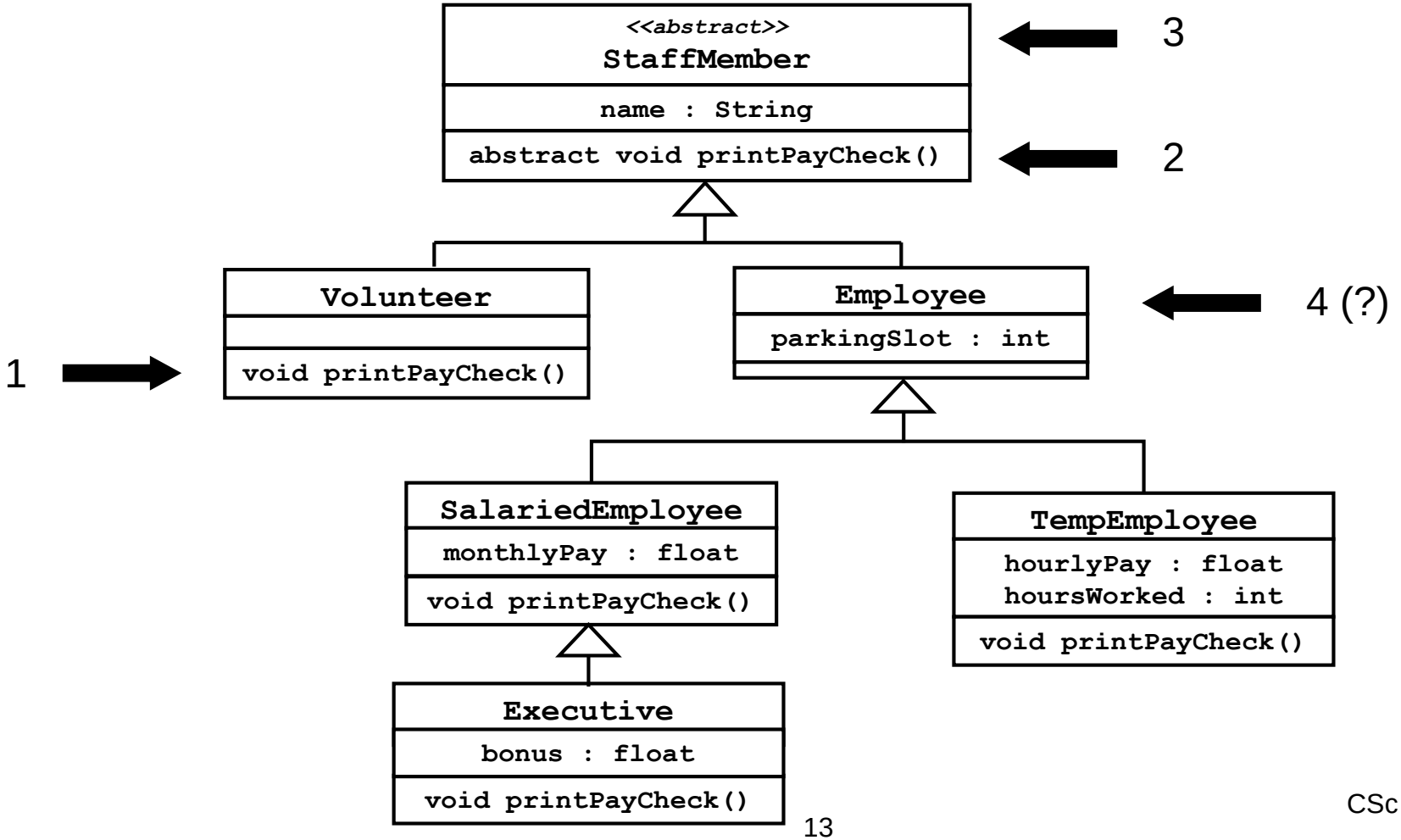
Now, the Print method which gets invoked is:

- *determined at runtime, and*
- *depends on subtype*

We still need to make sure it will compile, and that it is maintainable and extendable…

Polymorphic Safety

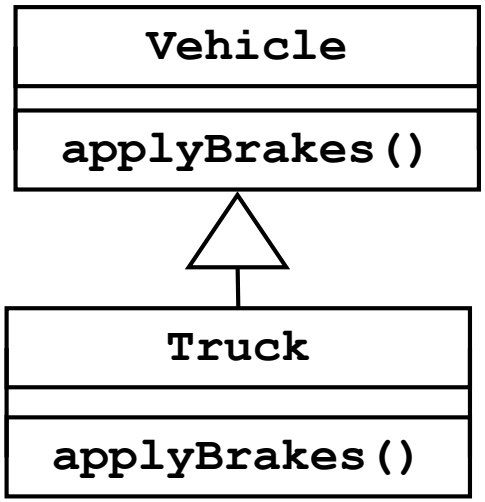
Ideally, *every* class should know how to deal with "**`printPayCheck`**" messages:



Polymorphism: Java

- **Java**
  - Run-time (dynamic; late) binding is the default
    - Drawback: may be unnecessary (hence inefficient)
    - Programmer can force compile-time binding by declaring methods "**`static`**, **`final`**, and/or **`private`** "

Java : Example



Java

```
class Vehicle {
  public void applyBrakes() {
    System.out.printf ("Applying vehicle brakes\n");
  }
}
class Truck extends Vehicle {
  public void applyBrakes() {
    System.out.printf("Applying truck brakes...\n");
  }
}
```

Java : Example (cont.)

Java

```
public static void main (String [] args){
  Vehicle v;
  Truck t;
  t = new Truck();
  t.applyBrakes();
  v = t ;
  v.applyBrakes();
}
```

Output

```
Applying truck brakes...
Applying truck brakes...
```

Polymorphism and Dynamic Binding

- If the object of the subclass has overridden a method in the superclass:
  - If the variable makes a call to that method, the subclass's version of the method will be run.

- Java performs *dynamic binding* or *late binding* when a variable contains a polymorphic reference.

- The Java Virtual Machine determines at runtime which method to call, depending on the type of object that the variable references.

Java : But #1, Can down-casting work? (cont.)

Java

```
public static void main (String [] args){
    Vehicle v2 = new Vehicle();
     Truck t2;
     t2 = (Truck) v2; // it realizes that Vehicle is not a Truck, it's only a Vehicle.
     t2.applyBrakes()
}
```

Output ?

Vehicle cannot be cast to Truck

We will discuss:

Apparent Type – as per program syntax (it is static and determined by thecompiler)

Actual Type – as per its creation (it is dynamic and determined during run time)

Java : But #2, Can down-casting work? (cont.)

Java

```
public static void main (String [] args){
     Truck t3 = new Truck();
     Vehicle v3 = t3;
     Truck t4 = (Truck) v3; // down casting
     t4.applyBrakes();
}
```

Output ?

```
Applying truck brakes...
```

Recap: Basis of Polymorphism

1. Inheritance
2. Method overriding
3. Polymorphic assignment // SuperClassVariable = SubclassObject;
4. Polymorphic methods
   - In Java, all methods are polymorphic.
   - That is, choice of method depends on the object.

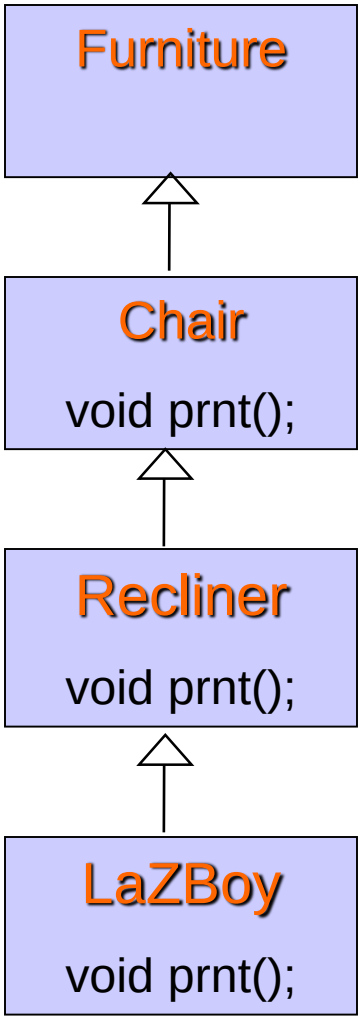
Quiz: Polymorphism

```
abstract class Furniture {
    public int numlegs;
}

abstract class Chair extends Furniture {
    public String fabric;
    abstract void prnt();
}

class Recliner extends Chair {
    void prnt() {
       System.out.println("I'm a recliner");
    }
}

class LaZBoy extends Recliner {
    void prnt() {
        System.out.println("I'm a lazboy");
    }
}
```



What is the output?

```
Chair cha;
cha = new LaZBoy();
cha.prnt();
```

```
Furniture furn;
furn = new Recliner();
furn.prnt();
```

```
Furniture furn;
furn = new LaZBoy();
furn.prnt();
```

Quiz: Members use compile-time binding

```
class Base{
        int X=99;
        public void prnt(){
                System.out.println("Base");
        }
}

class Rtype extends Base{
        int X=-1;
        public void prnt(){
                System.out.println("Rtype");
        }
}
```

What is the output?

```
Base b=new Rtype();
System.out.println(b.X);
b.prnt();
```