CSE 120/130

Introduction to

Programming Languages and Techniques

Fall 2000

Handout 9

Inheritance

The main idea of Java's inheritance mechanism is that a class can be defined by "extension" from another class. In the example below, `Employee` is declared as a class that extends the class `Person`. `Employee` will be also called a *subclass* of `Person` while `Person` would be called the `superclass` of `Employee`.

Note: In C++ `Employee` is called a *derived class* from the *base class* `Person`. Moreover, in C++ a class can be derived from several classes (multiple inheritance). Java has simple inheritance.

```
class Person {
private  int ss_num;

public String name;
public int birth_year;

public Person(int s, String n, int b){
  ss_num = s;
  name = n;
  birth_year = b;
}
public int get_ss(){
  return ss_num;
}
public int age(){
  return (Year.this_year - birth_year);
}
}
```



```
class Year{
  static int this_year = 1999;
}
```

- We made a class `Year` to hold the current year. Why?

Now let us build a subclass `Employee`

```
class Employee extends  Person{
 public int salary;       /* an additional data member */

 public  Employee(int ss, String n, int b, int sl){
   super(ss,n,b);         /* call constructor of superclass */
   salary = sl;           /* additional initialization */
 }
 public boolean is_yuppie(){         /* additional method */
   return (age() < 30 && salary > 80000);
 }
```


We can use the inherited class as follows:

```
public class Main{
  public static void main (String[] args){
    Person p = new Person(987654321, "Arthur Dent", 1955);
    System.out.println("Person name:   " +  p.name);
    System.out.println("Person ss_num: " + p.get_ss());

    Employee e = new Employee(123456789, "R. Rotwang", 1925, 40000);
    System.out.println("Employee name: " + e.name);
    System.out.println("Employee ss_num: " +  e.get_ss());
    System.out.println("Employee salary: " + e.salary);
    System.out.println("Employee yuppie? " +  e.is_yuppie());
  }
}
```

The public methods and data members of the class `Person` are inherited by the class `Employee`. The ouput is what we would expect. Note the appearance of a boolean constant.

```
Person name:   Arthur Dent
Person ss_num: 987654321
Employee name: R. Rotwang
Employee ss_num: 123456789
Employee salary: 40000
Employee yuppie? false
```


- As we mentioned, in Java a class extends exactly one other class. This produces a tree-shaped inheritance hierarchy. The root of this hierarchy is the class `Object`. If a class is declared without an `extends` clause then, by default, it is assumed to extend `Object`. Suprisingly, the class `Object` already has a number of methods which therefore apply to *all* Java objects.
- While multiple inheritance as in C++ is not possible, a related mechanism is achieved through *interfaces*; a class can implement several interfaces.

Overriding methods (late binding)

What happens when we have methods with the same type (and name) in two classes, one extending the other? Consider the following example.

```
class Person {
  String name;
  String address;
  Person(String n, String a) {
    name = n;
    address = a;
  }
  void printname(){System.out.println(name);}
  void printaddr(){
    printname();      /*Call to method in same class */
    System.out.println(address);
  }
}
```



```
class Lawyer extends Person{
  int sentence_length;

  Lawyer(String n, String a, int i){
    super(n,a);     /*Call the constructor of Person */
    sentence_length = i;
  }
  void printname(){
    System.out.println(name + ",  Esquire");
  }
}
```

```
class Knight extends Person{
  int waistline;

  Knight(String n, String a, int w){
    super(n,a);
    waistline = w;
  }
  void printname(){
    System.out.println("Sir " + name);
  }
}
```


Note that `printaddr` in class `Person` calls `printname` in the same class. Now suppose we call the `printaddr` method on a `Lawyer` object. The method `printaddr` is not defined in that class, so it calls the `printaddr` method in `Person`. But which `printname` method gets called? The answer is, for Java, the method in the class of the object – not the method of the class in which the calling method resides. To check this:

```
class Main{
  public static void main(String args[]){
    Person p =
      new Person("E. Bronte", "The Heights\nWuthering");
    Lawyer l =
      new Lawyer("Howe Dewey Cheatham",
                 "The Penitentiary\nMudville", 15);
    Knight k =
      new Knight("Topham Hat", "The Sidings\nKiddie City",56);
    p.printaddr();
    System.out.println();
    l.printaddr();
    System.out.println();
    k.printaddr();
    System.out.println();
  }
}
```


And this gives the output:

```
E. Bronte
The Heights
Wuthering

Howe Dewey Cheatham,  Esquire
The Penitentiary
Mudville

Sir Topham Hat
The Sidings
Kiddie City
```