Inheritance

November 12, 2007

ComS 207: Programming I (in Java)
Iowa State University, FALL 2007
Instructor: Alexander Stoytchev

Quick Review of Last Lecture

Passing Arguments

- Another important issue related to method design involves parameter passing
- Parameters in a Java method are *passed by value*
- A copy of the actual parameter (the value passed in) is stored into the formal parameter (in the method header)
- Therefore passing parameters is similar to an assignment statement

Passing Arguments

- Always done using “Pass By Value”

Example: PassByValue.java

Variable Assignment Revisited

- The act of assignment takes a copy of a value and stores it in a variable
- For primitive types:

Before:

`num1`	38
`num2`	96

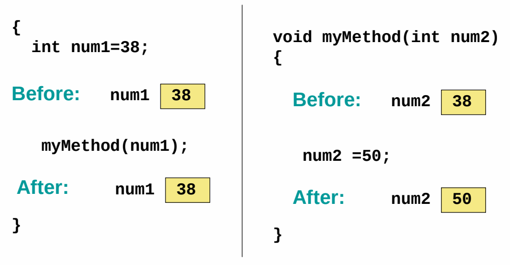
```
num2 = num1;
```

After:

`num1`	38
`num2`	38

Parameter Passing (primitive types)

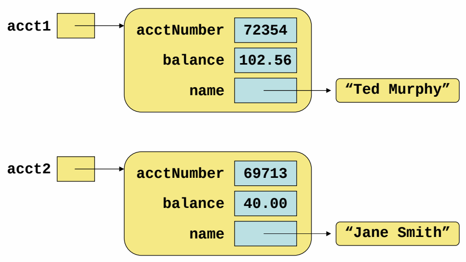
- The act of passing an argument takes a copy of a value and stores it in a local variable acessible only to the method which is being called.



```
{
  int num1=38;

   myMethod(num1);

}
```

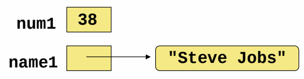
```
void myMethod(int num2)
{

  num2 =50;

}
```

Objects and Reference Variables



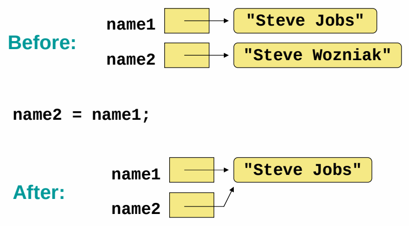
References

- Note that a primitive variable contains the value itself, but an object variable contains the address of the object
- An object reference can be thought of as a pointer to the location of the object
- Rather than dealing with arbitrary addresses, we often depict a reference graphically



Reference Assignment

- For object references, assignment copies the address:



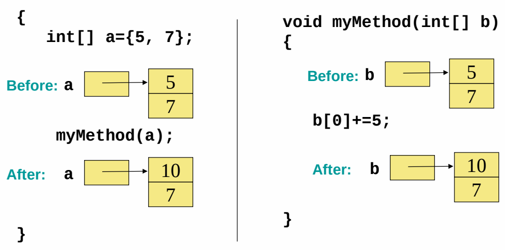
```
name2 = name1;
```

Aliases

- Two or more references that refer to the same object are called *aliases* of each other
- That creates an interesting situation: one object can be accessed using multiple reference variables
- Aliases can be useful, but should be managed carefully
- Changing an object through one reference changes it for all of its aliases, because there is really only one object

Parameter Passing (objects)

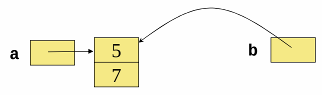
- Objects (in this case arrays) are also passed by value. In this case, however, the value is the address of the object pointed to by the reference variable.



```
{
   int[] a={5, 7};

     myMethod(a);

}
```

```
void myMethod(int[] b)
{

  b[0]+=5;

}
```

In the previous example there is only one array and two references to it.



The array can be modified through either reference.

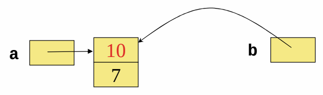
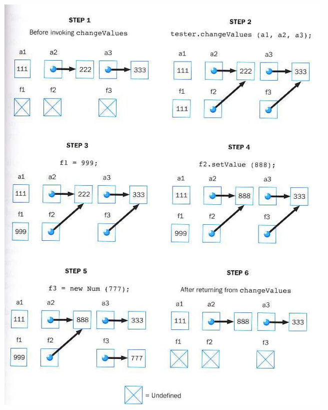


Figure 6.5



Objects as Parameters

- When an object is passed to a method, the actual parameter and the formal parameter become aliases of each other

Passing Objects to Methods

- What a method does with a parameter may or may not have a permanent effect (outside the method)
- See ParameterTester.java (page 327)
- See ParameterModifier.java (page 329)
- See Num.java (page 330)
- Note the difference between changing the internal state of an object versus changing which object a reference points to

Method Overloading

- *Method overloading* is the process of giving a single method name multiple definitions
- If a method is overloaded, the method name is not sufficient to determine which method is being called
- The *signature* of each overloaded method must be unique
- The signature includes the number, type, and order of the parameters

Method Overloading

- The compiler determines which method is being invoked by analyzing the parameters

```
float tryMe(int x)
{
    return x + .375;
}

float tryMe(int x, float y)
{
    return x*y;
}
```

Invocation

```
result = tryMe(25, 4.32)
```

Method Overloading

- The `println` method is overloaded:

```
println (String s)
println (int i)
println (double d)
```

and so on...

- The following lines invoke different versions of the `println` method:

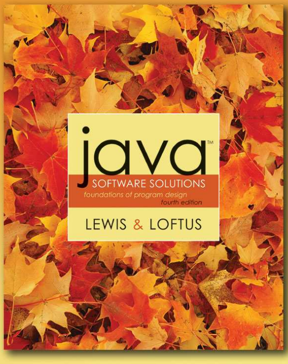
```
System.out.println ("The total is:");
System.out.println (total);
```

Overloading Methods

- The return type of the method is not part of the signature
- That is, overloaded methods cannot differ only by their return type
- Constructors can be overloaded
- Overloaded constructors provide multiple ways to initialize a new object

Chapter 8

Section 8.1



Inheritance

- Inheritance is a fundamental object-oriented design technique used to create and organize reusable classes
- Chapter 8 focuses on:
  - deriving new classes from existing classes
  - the `protected` modifier
  - creating class hierarchies
  - abstract classes
  - indirect visibility of inherited members
  - designing for inheritance
  - the GUI component class hierarchy
  - extending listener adapter classes
  - the `Timer` class

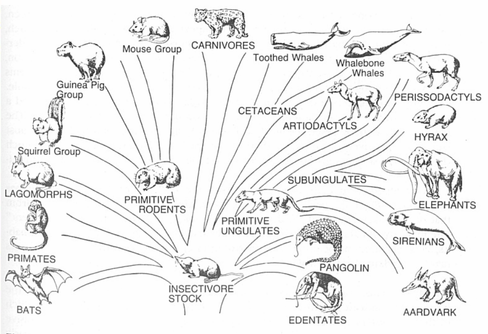
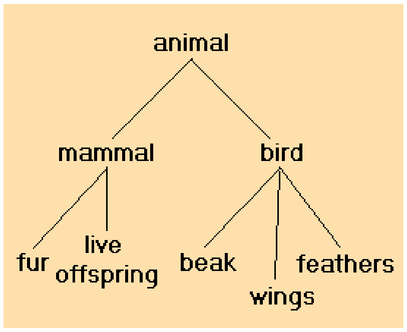


Figure 55. Diagrammatic family tree of the major orders (and some suborders) of eutherian (placental) mammals. Separate diagrams (Figs. 57–61) give in more detail the evolution of primates, carnivores, and odd- and even-toed ungulates.

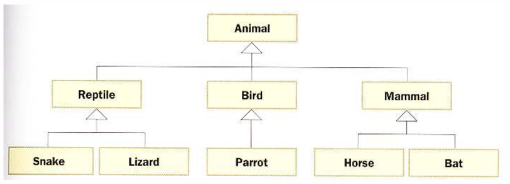
[http://cas.bellarmine.edu/tietjen/images/Mammal_order_tree.jpg]

Animals Class Hierarchy

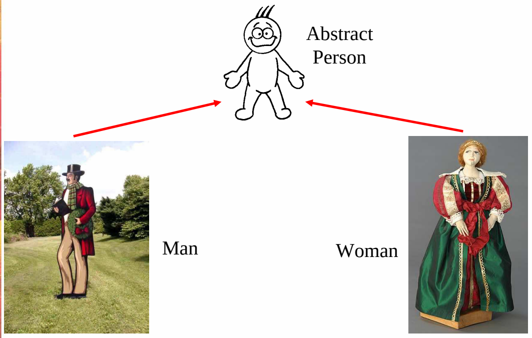


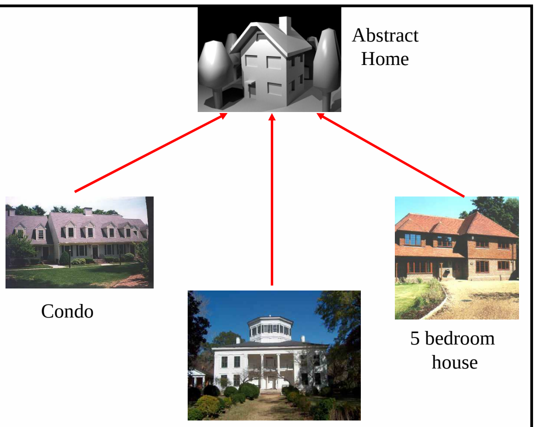
[http://www.scs.leeds.ac.uk/ugadmit/cogsci/tech/egtree.gif]

Animals Class Hierarchy



Inheritance Example



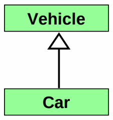


Inheritance

- *Inheritance* allows a software developer to derive a new class from an existing one
- The existing class is called the *parent class,* or *superclass*, or *base class*
- The derived class is called the *child class* or *subclass*
- As the name implies, the child inherits characteristics of the parent
- That is, the child class inherits the methods and data defined by the parent class

Inheritance

- Inheritance relationships are shown in a UML class diagram using a solid arrow with an unfilled triangular arrowhead pointing to the parent class



- Proper inheritance creates an *is-a* relationship, meaning the child *is a* more specific version of the parent

Inheritance

- A programmer can tailor a derived class as needed by adding new variables or methods, or by modifying the inherited ones
- *Software reuse* is a fundamental benefit of inheritance
- By using existing software components to create new ones, we capitalize on all the effort that went into the design, implementation, and testing of the existing software

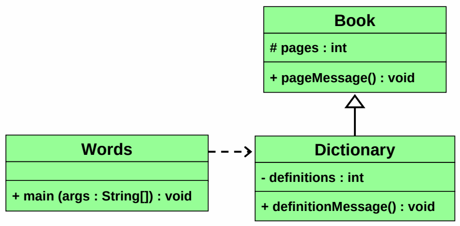
Deriving Subclasses

- In Java, we use the reserved word `extends` to establish an inheritance relationship

```
class Car extends Vehicle
{
   // class contents
}
```

- See `Words.java` (page 440)
- See `Book.java` (page 441)
- See `Dictionary.java` (page 442)

Class Diagram for Words



THE END