JAVA Programming

Reference Types

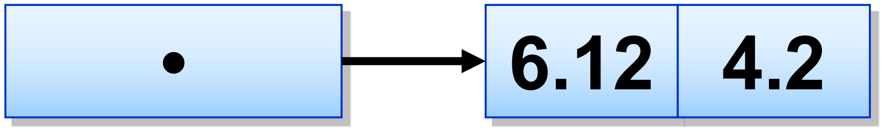
Overview

- Declaring and Releasing Reference Variables
- Comparing Values and Comparing references
- Multiple references to same object
- Using references as parameters
- The Object Hierarchy
  - The Object type
  - Common Methods
  - Conversion
  - Reflection

Declaring and Releasing Reference Variables

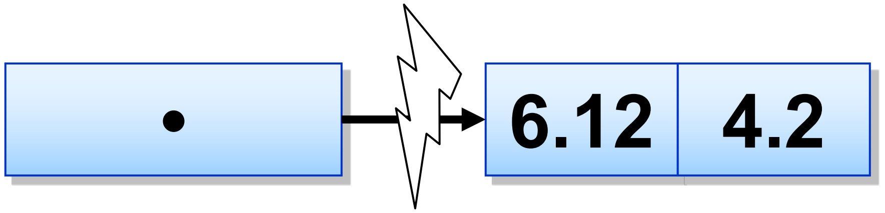
- Declaring reference variables

```
coordinate c1;
c1 = new coordinate();
c1.x = 6.12;
c1.y = 4.2;
```



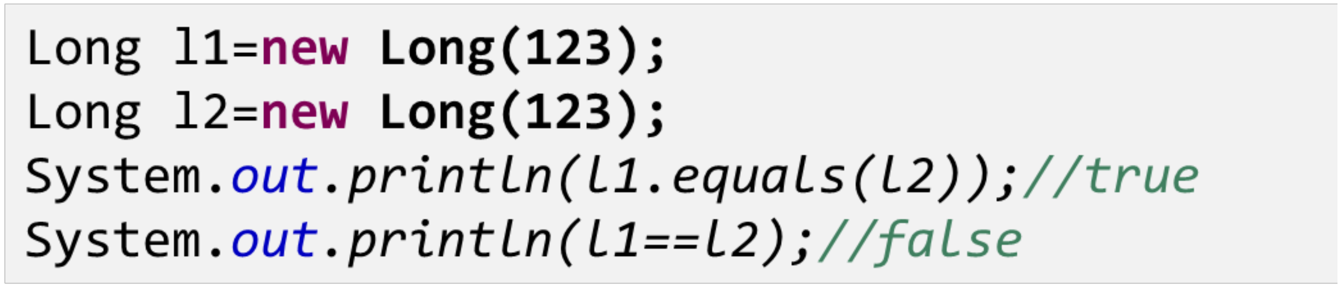
- Releasing reference variables

```
c1 = null;
```



Comparing Values and Comparing References

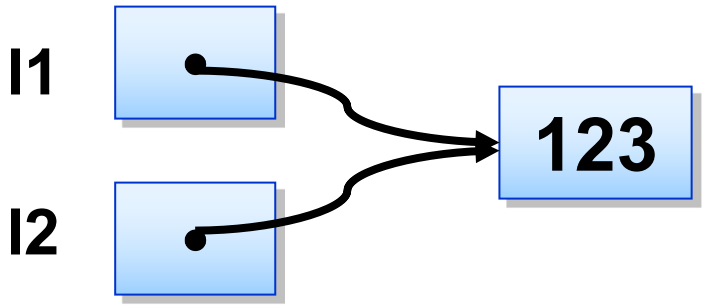
- Comparing primitive types
  - == and != compare values
- Comparing reference types
  - == and != compare the references, not the values
  - Use equals() to compare values



```
Long l1=new Long(123);
Long l2=new Long(123);
System.out.println(l1.equals(l2));//true
System.out.println(l1==l2);//false
```

Multiple References to the Same Object

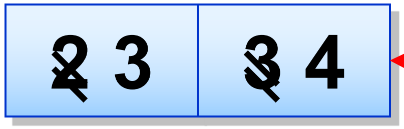
- Two references can refer to the same object
  - Two ways to access the same object for read/write



```
Long l1=new Long(123);
Long l2=l1;
System.out.println(l2);//123
System.out.println(l1.equals(l2));//true
System.out.println(l1==l2);//true
```

Using References as Method Parameters

- References can be used as parameters
  - When passed by value, data being referenced may be changed



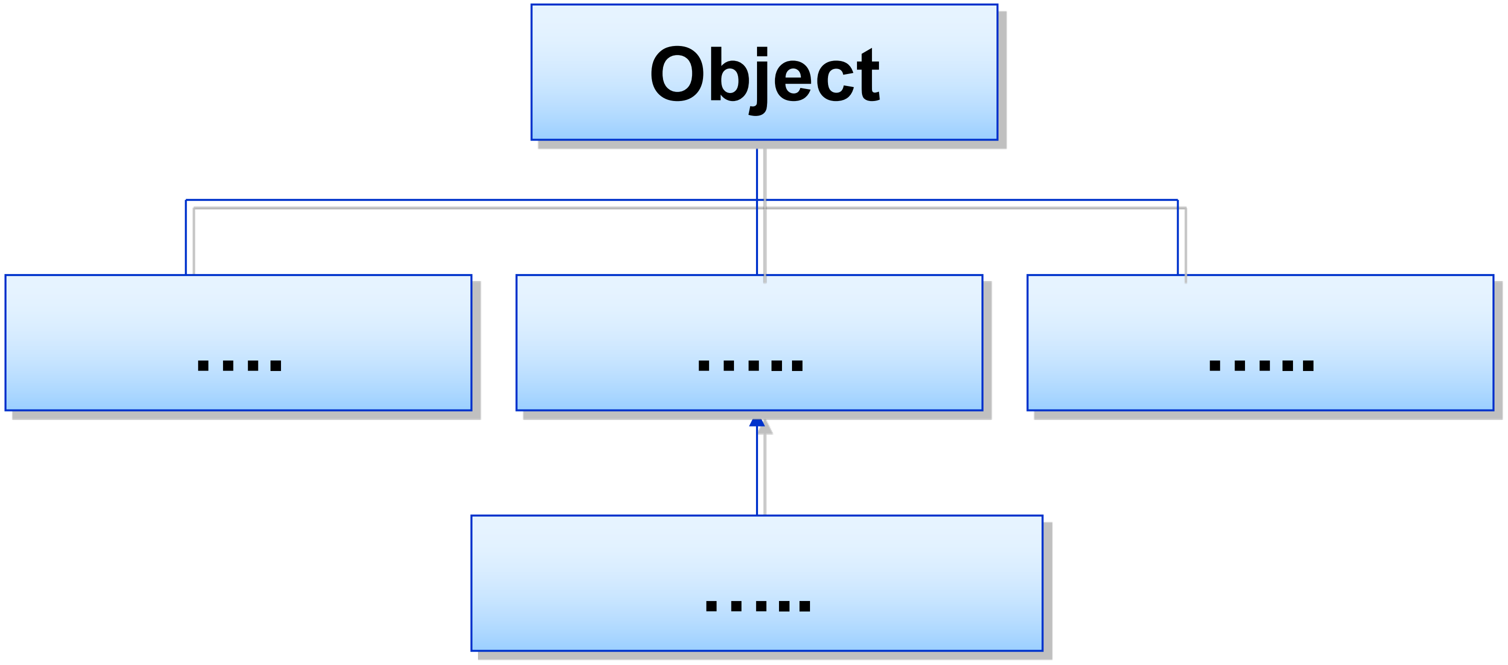
```
public static void main(String[] args) {
Coordinate co1=new Coordinate();
co1.setX(2);
co1.setY(3);
incrementCoordinate(co1);
System.out.println(co1);//Coordinate [x=3, y=4]
}
public static void incrementCoordinate(Coordinate c){
c.setX(c.getX()+1);
c.setY(c.getY()+1);
}
```

The Object Hierarchy

- The Object Type
- Common Methods
- Conversion
- Reflection

The Object Type

- Synonym for java.lang.Object
- Base class for all classes



Common Methods

- Common methods for all reference types
  - **toString** method
  - **equals** method
  - **clone**method
  - **finalize** method
  - **getClass** method
  - **hashCode** method
  - **notify** method
  - **notifyAll** method
  - **wait** method

Converting Value Types

- Implicit conversions
- Explicit conversions
  – Cast operator
- Exceptions

Parent/Child Conversions

- Conversion to base class reference
  - Implicit or explicit
  - Always succeeds
  - Can always assign to object
- Conversion to derived class reference
  - Explicit casting required
  - Will check that the reference is of the correct type
  - Will raise **ClassCastException** if not

Base class / Derived class Conversions

```
Person person;
Employee employee = new Employee();
person = employee;// implicit cast

Person[] persons = new Person[10];
// ..
if (persons[2] instanceof Employee) {
  // person in third cell is an employee
  System.out.println(((Employee) persons[2]).getEmpID());
  //explicit cast
}
```

Conversions and the object Type

- The Object type is the base for all classes
- Any reference can be assigned to Object
- Any Object variable can be assigned to any reference
  - With appropriate type conversion and checks

```
Person p= new Person()
Object ox;
ox = p;
ox = (Object) p;
```

```
p = (Person) ox;
```

Reflection

- Classes from the reflection package can be used to examine a type in detail.
- Start point is a Class object, from where you can obtain
  - Complete list of members
  - Implemented interfaces
  - Classes it extends
  - Modifiers applied
  - metadata

Lab

No lab associated with this module