COMP112/18 - Programming I

18 Classes and Objects (2)

Instructor: Ke Wei（柯韋）

A319 ✆ Ext. 6452 ✉ wke@ipm.edu.mo

http://brouwer.ipm.edu.mo/COMP112/18/

Bachelor of Science in Computing, School of Public Administration, Macao Polytechnic Institute

November 16, 2018

Outline

Passing and Returning Objects

- When we pass and return an object to and from a method, we transfers only the reference to the object.
- The method below returns a copy of a rectangle.

```
public static Rectangle copyRectangle(Rectangle r) {
    Rectangle q = new Rectangle();
    q.width = r.width;
    q.height = r.height;
    return q;
}
```

- The following method returns `true` only if the first circle is smaller than the second circle.

```
public static boolean lessThan(Circle a, Circle b) {
    return a.radius < b.radius;
}
```

Example: Planar Vectors

```
1  public class Vec {
2      public double x, y;
3
4      public Vec() { x = 0.0; y = 0.0; }
5      public Vec(double x, double y) { this.x = x; this.y = y; }
6      public Vec neg() { return new Vec(-x, -y); }
7      public Vec add(Vec v) { return new Vec(x + v.x, y + v.y); }
8      public double dot(Vec v) { return x * v.x + y * v.y; }
9      public double len(Vec v) { return Math.sqrt(dot(this)); }
10     public Vec sca(double r) { return new Vec(x * r, y * r); }
11     public Vec rot(double r) {
12         double s = Math.sin(r), c = Math.cos(r);
13         return new Vec(x * c - y * s, x * s + y * c);
14     }
15 }
```


Using Vectors

- Given three points $P=(1,2)$, $A=(2,5)$ and $B=(-1,7)$, compute the area of $\triangle APB$. Let $\vec{a}=\overrightarrow{PA}$ and $\vec{b}=\overrightarrow{PB}$. We compute the area by $\frac{\sqrt{(\vec{a}\cdot\vec{a})(\vec{b}\cdot\vec{b})-(\vec{a}\cdot\vec{b})^2}}{2}$.

  ```
  Vec P = new Vec(1, 2), A = new Vec(2, 5), B = new Vec(-1, 7);
  Vec a = A.add(P.neg()), b = B.add(P.neg());
  System.out.println(Math.sqrt(a.dot(a)*b.dot(b) - a.dot(b)*a.dot(b)) / 2);
  ```

- Given two points $Q=(3,5)$ and $K=(10,7)$, find the point of rotating K about Q by $-90°$. We rotate vector $\overrightarrow{KQ}$ and move it back with offset $\vec{Q}$.

  ```
  Vec Q = new Vec(3, 5), K = new Vec(10, 7);
  Vec k = K.add(Q.neg()), j = k.rot(Math.toRadians(-90)).add(Q);
  System.out.println("(" + j.x + "," + j.y + ")");
  ```

- The methods defined in *Vec* do not change `this` object, instead, they create new objects as the results of the operations. Objects of such a class are called *immutable*.

Showing an Object as a String

- An object is automatically converted to a string when being printed or concatenated with another string.
- This conversion is performed through the predefined method *toString* in the *Object* class.
- However, the predefined method does not really know how to show an object of your class as a string.
- You may *override* the *toString* method to define your way of showing your class objects.

  ```
  public class Vec { ...
      @Override public String toString() {
          return "Vec(x="+this.x+", y="+this.y+")";
      }
  }
  ```

- Then, *System.out.print*(`"K = "+K`); shows `"K = Vec(x=10.0, y=7.0)"`.

The *Object* Class

- Every class in Java, user defined or system defined, is a *subclass* of the *Object* class.
- The *Object* class is the *superclass* of all classes.
- By the Java object-oriented mechanism, an object of a subclass is also an object of any of the superclasses.
- Therefore, all objects in Java are of the *Object* class. A variable of *Object* can store a reference to an object of any class.

```
Object x = new int[5], y = new Vec(3, 4);
```

- However, turning a reference to *Object* back to a reference of some other class requires an explicit cast.

```
int[] a = (int[]) x; Vec v = (Vec) y;
```

- We have a way to define an array of any type of data — with element type *Object*.

Autoboxing and Unboxing of Primitive Types

- Although objects of any class can be assigned to variables of class *Object*, we still have values of primitive types, such as `int` and `boolean`.
- Primitive type values are stored directly in variables, they are not accessed through references.
- To make the *Object* class truly universal, we must have a way to *wrap* a primitive value into an object. This is called *boxing*.

```
class Integer { int value; }
class Boolean { boolean value; }
```

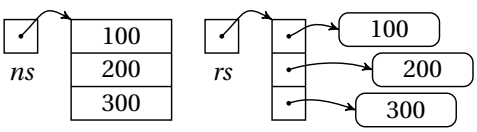
- Objects of such classes can then be assigned to variables of the *Object* class.
- The system automatically boxes a primitive type value when it is used as an object, and unboxes an object when it is used as a primitive type value.

```
Integer a = 10;    // a = new Integer(10);
int b = a + 20;    // b = a.intValue() + 20;
```


Autoboxing Illustrated

When an `int` value is assigned to an *Object* variable, the value is boxed in an *Integer* object, and the reference to the *Integer* object is assigned as a reference to *Object*.

```
int[] ns = new int[] {100, 200, 300};
Object[] rs = new Object[ns.length];
for ( int i = 0; i < ns.length; ++i )
    rs[i] = ns[i];
```



Each primitive data type in Java has a corresponding wrapper class, listed below.

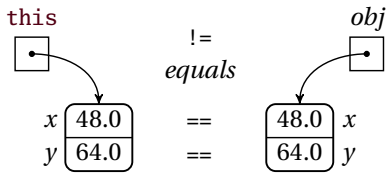
`boolean`	*Boolean*
`byte`	*Byte*
`char`	*Character*
`float`	*Float*

`int`	*Integer*
`long`	*Long*
`short`	*Short*
`double`	*Double*

Shallow Equality and Content (Deep) Equality

- The built-in equality operator "==" compares shallowly. If the two operands are references, they must point to the same object to be considered equal.
- You must override the "*equals*" method to perform you own content equality tests.

```
public class Vec { ...
    @Override
    public boolean equals(Object obj) {
        if ( obj != null && getClass() == obj.getClass() ) {
            Vec v = (Vec)obj;
            return x == v.x && y == v.y;
        } else return false;
    }
}
```



Deep Equality

If the fields are themselves references, you may need to propagate the equality tests "deeper" down to the instances being referred to.

```
class Triangle {
    Vec a, b, c; ...
    @Override
    public boolean equals(Object obj) {
        if ( obj != null && getClass() == obj.getClass() ) {
            Triangle tri = (Triangle)obj;
            return a.equals(tri.a) && b.equals(tri.b) && c.equals(tri.c);
        } else return false;
    }
}
```

✍ Try to illustrate the deep equality test of two triangles.

Reading Homework

Textbook

- Section 9.10–9.11.
- Section 10.3–10.4, 10.7–10.9.
- Section 11.1–11.2, 11.6, 11.9–11.10.

Internet

- Polymorphism (`http://en.wikipedia.org/wiki/Polymorphism_(computer_science)`).