17 Classes and Objects

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 12, 2018

Outline

Object Oriented (OO) Programming Concepts

- Object-oriented programming (OOP) involves programming using objects.
- An *object* represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
- An object has a unique identity, state, and behaviors.
- The state of an object consists of a set of variables, called *data fields* (also known as properties, attributes or *instance variables*) with their current values.
- The behavior of an object is defined by a set of methods.

Classes and Objects

A class template:

Class Name: *Rectangle*
Data Fields:
  width is ________
  height is ________
Methods:
  getArea

Three objects of the *Rectangle* class:

Rectangle
Object 1
Data Fields:
  width is 10
  height is 5.5

Rectangle
Object 2
Data Fields:
  width is 16
  height is 10

Rectangle
Object 3
Data Fields:
  width is 40
  height is 30

- *Classes* are constructs that define objects of the same type, including the layout of the data fields and the definition of the methods.
- Objects of the same class each have their own *instances* of data fields, but share the same definition of the methods.
- Additionally, a class provides a special type of methods, known as *constructors*, which are invoked to construct objects from the class.

A Class Example

```
class Rectangle {
    // data fields
    double width, height;

    // constructors
    Rectangle() { width = 1.0; height = 1.0; }

    Rectangle(double width, double height) {
        this.width = width;   // Local variables hide the fields with the same names.
        this.height = height;
    }

    // method
    double getArea() { return width * height; }
}
```

Constructors

- Constructors are a special kind of methods that are invoked to initialize objects.
- A constructor with no parameters is referred to as a *no-arg constructor*.

```
Rectangle() { width = 1.0; height = 1.0; }
```

- Constructors must have the same name as the class itself.
- Multiple constructors can be defined as long as they take different types of parameters.

```
Rectangle(double width, double height) {
    this.width = width;   // Local variables hide the fields with the same names.
    this.height = height;
}
```

- Constructors do not have a return type — not even `void`.
- Constructors are invoked using the `new` operator when an object is created.

```
Rectangle a = new Rectangle();            // a 1.0 × 1.0 rectangle
Rectangle b = new Rectangle(10.0, 5.5);   // a 10.0 × 5.5 rectangle
```

Default Constructor

- A class may be defined without constructors.

```
class Circle {
    double radius = 1.0;
    double getArea() { return radius * radius * Math.PI; }
}
```

- In this case, a no-arg constructor with an empty body is implicitly declared for the class.

```
Circle c = new Circle();    // a circle with radius 1.0
```

- This constructor, called a *default constructor*, is provided automatically *only if NO constructors are explicitly defined* in the class.

```
class CircleWithCons {
    double radius;
    CircleWithCons(double radius) { this.radius = radius; }
} ...
CircleWithCons d = new CircleWithCons(); // ✗ WRONG! No default constructor.
```

Accessing Object's Fields via Reference Variables

- We must use an object through a reference.
- References are generated by the `new` operation, they can be passed to and returned from methods, and they can be stored in reference variables.
- To declare a reference variable, use the syntax: $\boxed{\textit{ClassName objectRefVar};}$

```
Circle myCircle;
```

- Referencing the object's data field: $\boxed{\textit{objectRefVar}.\textit{field}}$

```
myCircle.radius = 10.0;
System.out.println(myCircle.radius);
```

- Invoking the object's method: $\boxed{\textit{objectRefVar}.\textit{methodName}(\textit{arguments})}$

```
System.out.println(2 * myCircle.getArea());
```

Variables of Primitive Data Types and Reference Types

- Variables of primitive data types store values directly, these types include

  `int`, `byte`, `short`, `long`, `boolean`, `char`, `double` and `float`

  They are also called value types.
- Variables of reference types store references (pointers) to objects, these types are system and user defined classes. They are also called object or reference types.
- *String* is a system defined class, so variables of *String* are references.
- Assignments to value type variables copy the values.
- Assignments to reference type variables copy the references, but not the objects.
- Two reference variables are equal only if they point to the same object.

  $$String\ a = \texttt{new}\ String(\texttt{"ABC"}),\ b = a,\ c = \texttt{new}\ String(\texttt{"ABC"});$$

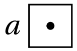
  We have $a == b$ but $a\ != c$. However, $a.equals(c)$ returns `true`.
- References returned by the `new` operator are different from all existing references.

Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```

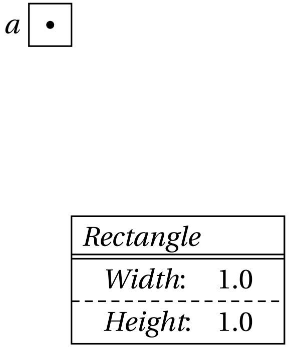
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



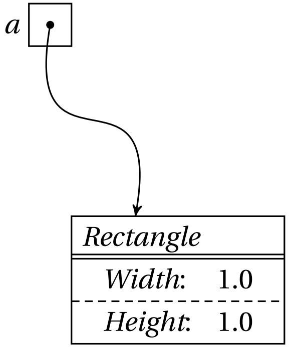
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



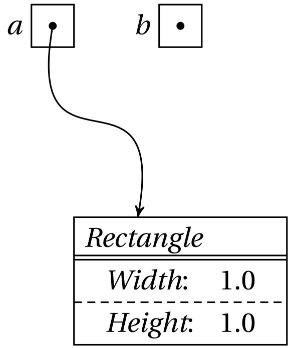
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



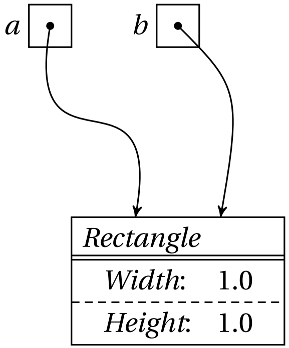
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



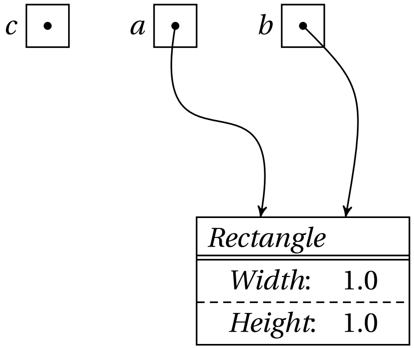
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



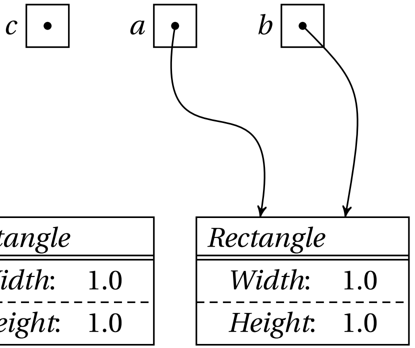
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



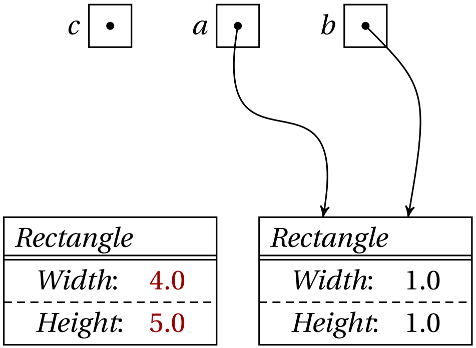
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



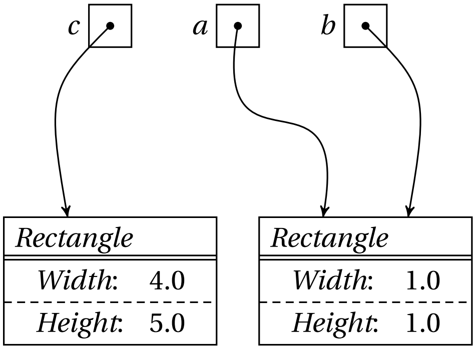
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



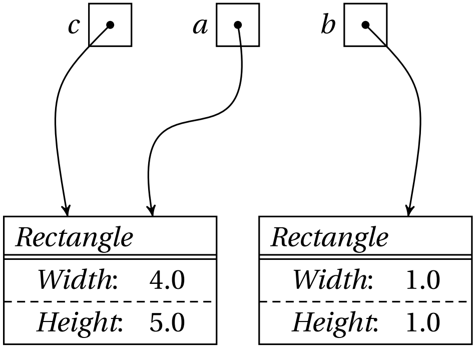
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



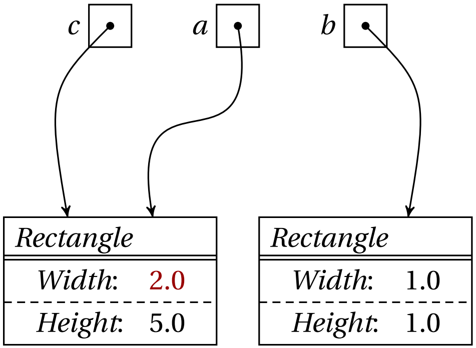
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



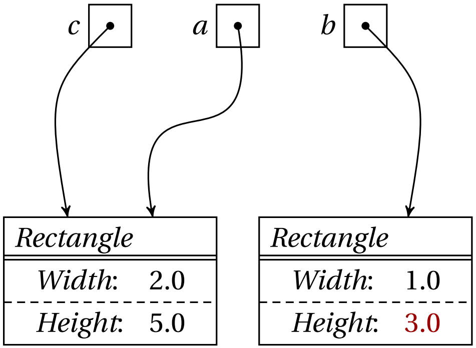
Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



Tracing Object Creations and Assignments

```
Rectangle a = new Rectangle(), b = a;
Rectangle c = new Rectangle(4, 5);
a = c;
a.width = 2;
b.height = 3;
```



Practice: Drawing the Memory Diagram

Given the *Rec* class defined below.

```
public class Rec {
    char id;
    int value;
    public Rec(char id, int value) { this.id = id; this.value = value; }
}
```

Draw the memory diagram after the execution of the following code.

```
Rec a = new Rec('a', 20), b = new Rec('b', 30), c = new Rec('c', 10);
Rec t = a;
a = b;
b = c;
c = t;
a.value += b.value;
c.value *= a.value;
```

Reading Homework

Textbook

- Section 9.1–9.5.

Internet

- Object-oriented programming (`http://en.wikipedia.org/wiki/Object-oriented_programming`).
- Object (`http://en.wikipedia.org/wiki/Object_(object-oriented_programming)`).
- Class (`http://en.wikipedia.org/wiki/Class_(computer_programming)`).
- Reference (`http://en.wikipedia.org/wiki/Reference_(computer_science)`).