Class & Object

Tanjina Helaly

Two Paradigms in OOP

- All computer programs consist of two elements:
  - code and data.
- A program can be conceptually organized around its code or around its data.
- That is, some programs are written around "what is happening" and others are written around "who is being affected."
- These are the two paradigms that govern how an OOP program is constructed.
- In Procedural language data and operation/code are separate
- Example (using C)
  - Student – 4 fields: name, id, cgpa, creditCompleted
  - Update cgpa – need a function to calculate the new cgpa

Class & Object

- Object
  - An object is a software bundle of related state and behavior.
  - Software objects are often used to model the real-world objects that you find in everyday life.

- Class
  - A class is a blueprint or prototype from which objects are created.

What is Class

- the class is the basis for object-oriented programming in Java.
- it defines a new data type.
- Once defined, this new type can be used to create objects of that type.
  - Thus, a class is a *template for an object,*
  - *and an object is an instance of a class.*
  - *Because an object is an* instance of a class, you will often see the two words *object and instance used interchangeably*

Class Members

- Class contains the following 2 members
  - instance variables(properties/attributes)
    - **each instance of the class** (that is, each object of the class) **contains its own copy of these variables**.
    - Thus, the data for one object is separate and unique from the data for another.
  - Methods(action/behavior)
    - *The code is* contained within *methods*
    - In most classes, the instance variables are acted upon and accessed by the methods defined for that class.
    - Thus, as a general rule, it is the methods that determine how a class' data can be used.

The General Form of a Class

```
class classname {
    datatype instance-variable1;
    datatype instance-variable2;
    // ...
    datatype instance-variableN;

    return-type methodname1(parameter-list) {
    // body of method
    }

    return-type methodname2(parameter-list) {
    // body of method
    }

    // ...

    return-type methodnameN(parameter-list) {
    // body of method
    }
}
```

A Simple Class

```
public class Student{
    // Instance variables
    public String name;
    public String id;
    public float cgpa;
    public int creditCompleted;

    // Methods
    public void updateCgpa(int credit, float gpa){
        cgpa = (cgpa*creditCompleted + credit*gpa)/(creditCompleted+credit);
        creditCompleted = creditCompleted + credit;
    }

    public float getCgpa(){
        return cgpa;
    }
}
```

Create Object and access Members

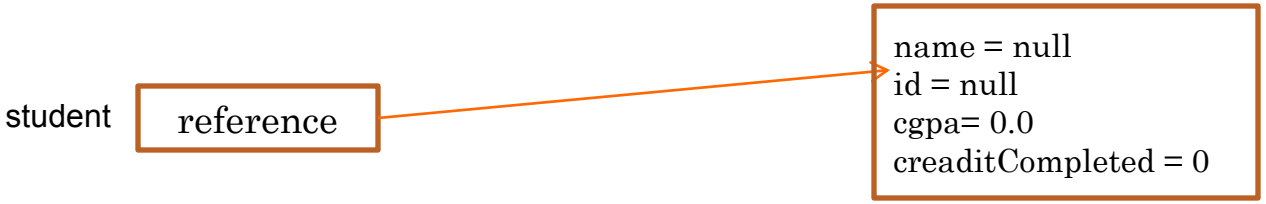
```
public class TestStudent {

    public static void main(String[] args) {
        // Create object
        Student student = new Student();

        // Assign values to attributes
        student.name = "Rashid";
        student.id = "011153001";
        student.cgpa = 3.0f;
        student.creditCompleted = 50;

        // display cgpa before update
        System.out.println("Credit Completed: "+ student.creditCompleted +"; Previous cgpa: "+ student.cgpa);

        // Calling method - update cgpa
        student.updateCgpa(3, 4.0f);
        student.updateCgpa(3, 4.0f);
        student.updateCgpa(3, 4.0f);

        // display cgpa after update
        System.out.printf("Credit Completed: %d; New cgpa: %.2f", student.creditCompleted,student.cgpa);
    }
}
```

Output:
Credit Completed: 50; Previous cgpa: 3.0
Credit Completed: 59; New cgpa: 3.15

Reference Variable

Student student;

student = **new Student();**

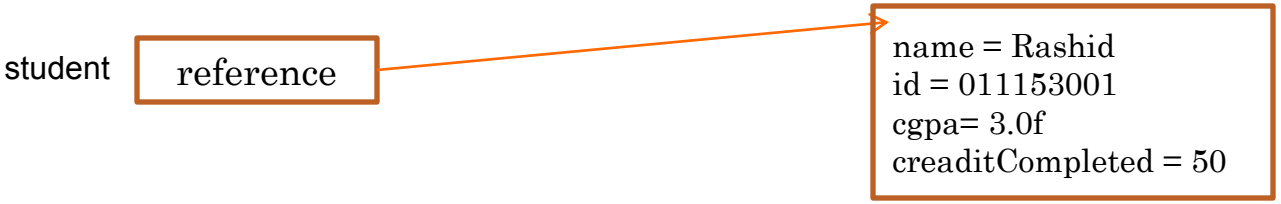


student.name = “Rashid”;

student.id = “011153001”;

student.cgpa = 3.0f;

student.creditCompleted = 50;



Create Object and access Members

```
public class TestBankAccount {
    public static void main(String[] args) {
    // Create object
    Student studentR = new Student();
    Student studentK = new Student();

    // Assign values to attributes
    studentR.name = "Rashid";  studentR.cgpa = 3.0f; studentR.creditCompleted = 20;

    studentK.name = "Khaled"; studentK.cgpa = 3.0f; studentK.creditCompleted = 20;

    // display cgpa before update
    System.out.println(studentR.name + "; Credit Completed: "+ studentR.creditCompleted +"; Previous
    cgpa: "+ studentR.cgpa);
    System.out.println(studentK.name + "; Credit Completed: "+ studentK.creditCompleted +"; Previous
    cgpa: "+ studentK.cgpa);

    // update cgpa
    studentR.updateCgpa(3, 4.0f);

    // display cgpa after update
    System.out.println("After Update");
    System.out.printf("%s; Credit Completed: %d; New cgpa: %.2f\n", studentR.name,
    studentR.creditCompleted, studentR.cgpa);
    System.out.printf("%s; Credit Completed: %d; New cgpa: %.2f", studentK.name,
    studentK.creditCompleted, studentK.cgpa);
    }
}
```

Constructor

- A *constructor*
  - *Allocate space for instance variables.*
  - *initializes an object(its instance variables) immediately upon creation.*
- *Syntax:*
  - *It has the same name as* the class.
  - syntactically similar to a method.
    - Except has no return type. Not even **void**.
      - **This is because the implicit return type of a class' constructor is the class type itself.**

Constructor

- When called:
  - No explicit call
  - It is automatically called when the object is created, before the **new operator** completes.
- What should go inside Constructor
  - Normally the instance variables are initialized inside the constructor.

    Or
  - any set-up code

Constructor - Example

```
import java.util.Random;
public class BankAccount {
    // Instance variables
    public String name;
    public String id;
    public double balance;

    // Constructor without parameter
    public BankAccount(){
        id = new Random().nextInt(99999) + "";
        // name and balance will get default value
    }

    // Constructor with parameter
    public BankAccount(String _name, String _id, double _balance){
        name = _name;
        id = _id;
        balance = _balance;
    }

    public static void main(String[] args)
    {
        BankAccount ba = new BankAccount("Rashid", "1000500", 1000.0);
    }
}
```

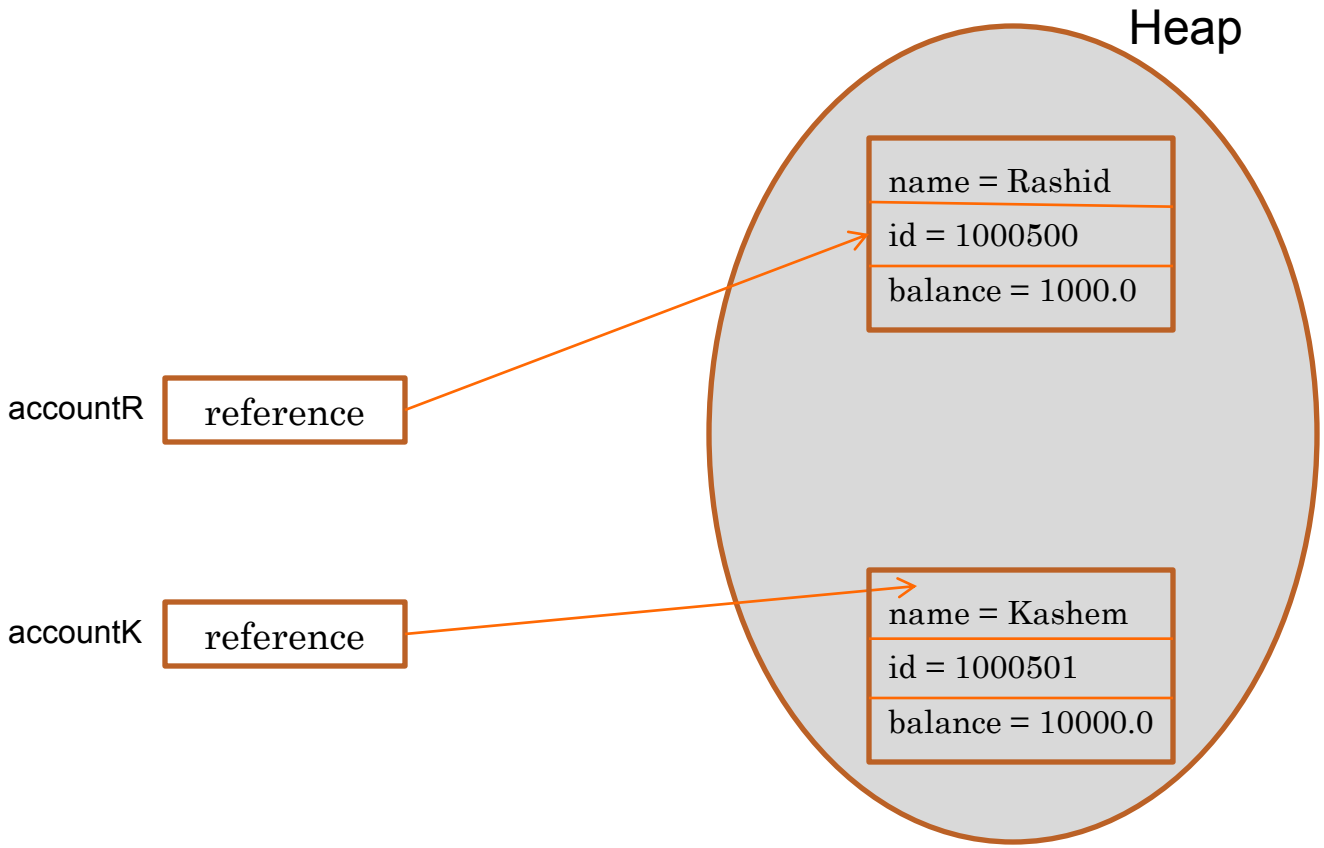
Default Constructor

- When you **do not explicitly define a constructor** for a class,
  - then Java **creates a default constructor** for the class.
  - This is why the first BankAccount class code worked even though **that did not define a constructor.**

- **The default** constructor automatically **initializes all instance variables to their default values**,
  - Default value
    - Boolean -> false;
    - All primitive except boolean -> 0 (zero)
    - Reference type -> null
- Once you define your own constructor, the default constructor is no longer used.

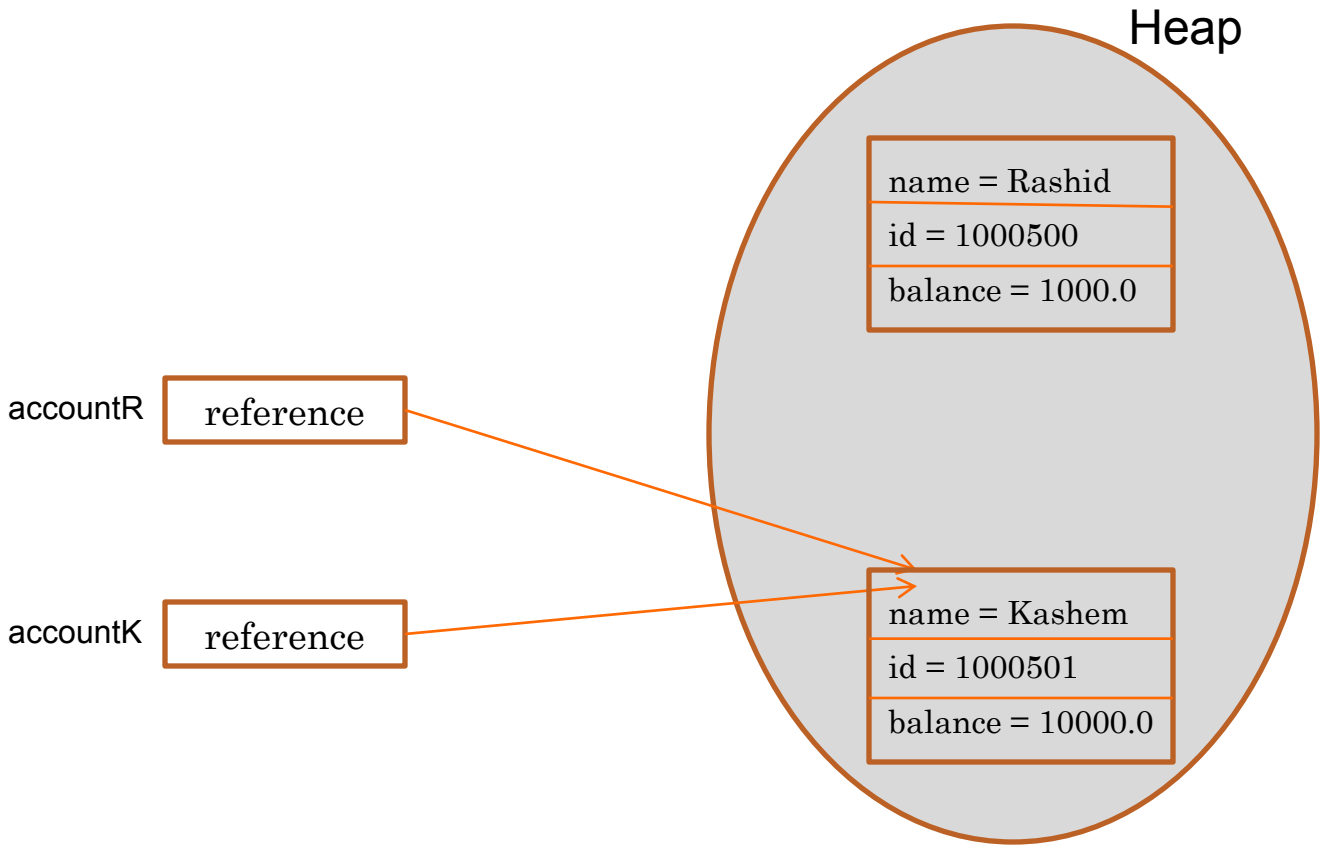
Garbage Collection

BankAccount accountR = **new BankAccount("Rashid", "1000500" , 1000.0);**
BankAccount accountK = **new BankAccount(("Kashem", "1000501" , 10000.0);**



Garbage Collection

BankAccount accountR = **new BankAccount(“Rashid”, “1000500” , 1000.0);**
BankAccount accountK = **new BankAccount((“Kashem”, “1000501” , 10000.0);**
accountR = accountK;



Reference

- Java:Complete Reference Chapter 6