Objects and Classes

Part 7 – More Class Examples

Chapter 4, Core Java, Volume I

Contents

- Class Examples
- Aggregation Relationship
- Example: Extending Employee with Account
- Data Classes
- Data Class Examples

Class Example : Account Class

```
// Account.java
public class Account
{
   private final String name;
   private double balance;
   public Account(String name, double balance)
   {
      this.name = name;
      if (balance >= 0.0) // if the balance is valid
         this.balance = balance;
      else
         this.balance = 0;
   }
   public Account(String name) // overloaded
   {
      this(name, 0.0);  // calls Account(String, double)
   }
```

```
   public void deposit(double depositAmount)
   {
      if (depositAmount > 0.0)
         balance = balance + depositAmount;
   }
   public double getBalance() // getter
   {
      return balance;
   }
   public String getName() // getter
   {
      return name;
   }
} // end of Account
```

Class Example : Account Class

```
// AccountTest.java
import java.util.Scanner;
public class AccountTest
{
   public static void main(String[] args)
   {
      Account account1 = new Account("Jane Green", 50.00);
      Account account2 = new Account("John Blue");
      System.out.printf("%s balance: $%.2f%n",
            account1.getName(), account1.getBalance());
      System.out.printf("%s balance: $%.2f%n%n",
            account2.getName(), account2.getBalance());
      Scanner input = new Scanner(System.in);
      System.out.print("Enter deposit amount for account1: ");
      double depositAmount = input.nextDouble();
      account1.deposit(depositAmount);
      System.out.printf("%s balance: $%.2f%n",
            account1.getName(), account1.getBalance());
      System.out.printf("%s balance: $%.2f%n%n",
            account2.getName(), account2.getBalance());

      Account account3 = new Account("Mary Red, -75.0);
      System.out.printf("%s balance: $%.2f%n%n",
            account3.getName(), account3.getBalance());
   } // end main
} // end class AccountTest
```

Class Example : Stack Class

```
// Stack.java
public class Stack
{
   private int top;
   private int[] stack;
   private int capacity;

   public Stack(int capacity)
   {
      top = -1;
      if (capacity <= 0) // validation check
         capacity = 10;
      else
         this.capacity = capacity;
      stack = new int[capacity];
   }
```

```
   public void push(int d)
   {
      top++;
      if(isFull()) {
         System.out.println("Stack Full");
         System.exit(-1);
      }
      stack[top]=d;
   }
   private boolean isFull() // helper function; not public
   {
      if(top>=capacity)
         return true;
      else
         return false;
   }
```

Class Example : Stack Class

```
  public int pop()
  {
    if(isEmpty()) {
        System.out.println("Stack Empty");
        System.exit(-1);
    }
    top--;
    return stack[top+1];
  }
  private boolean isEmpty()
  {
     if(top<0)
       return true;
     else
       return false;
  }
} // end of Stack
```

```
// StackTest.java
public class StackTest
{
  public static void main(String[] args)
  {
      Stack s = new Stack(10);

      s.push(1);
      s.push(2);
      System.out.println(s.pop());
      s.push(3);
      System.out.println(s.pop());
      System.out.println(s.pop());
  } //end of main
```

Class Example : Stack Class (Simple Unit Test)

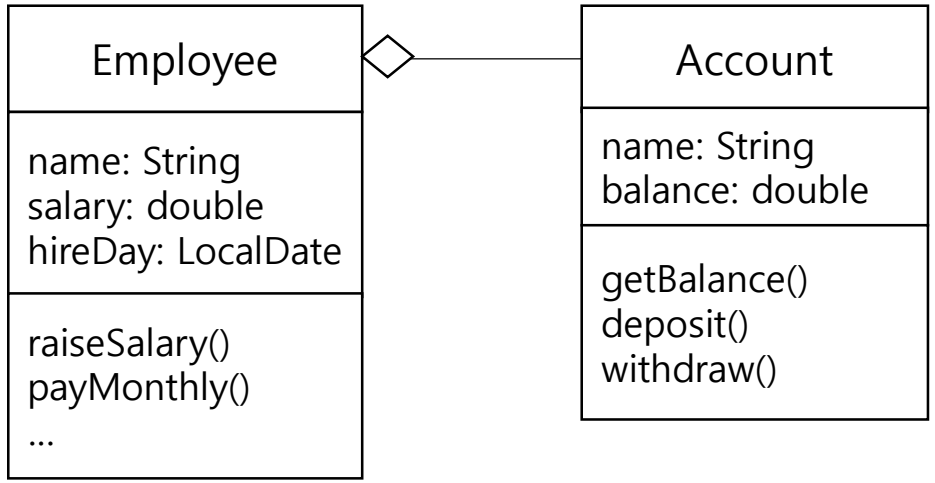
```
 // Stack.java
public class Stack
{
   private int top;
   private int[] stack;
   private int capacity;
   public Stack(int capacity)
   {
      top = -1;
      if (capcity <= 0) // validation check
         capacity = 10;
      else
         this.capacity = capacity;
     stack = new [capacity];
   }
  // push …
  // pop …
```

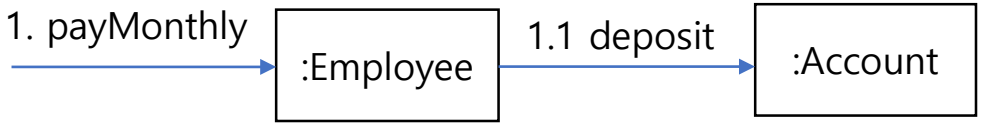
```
   public static void main(String[] args)
   {
        Stack s = new Stack(10);

        s.push(1);
        s.push(2);
        System.out.println(s.pop());
        s.push(3);
        System.out.println(s.pop());
        System.out.println(s.pop());
   } //end of main
} // end of Stack
```

Aggregation Relationship (or Composition Relationship)

- Part-whole (has-a) relationship
- Composition relationship – stronger part-whole relationship

- Extending Employee with Account





Example: Extending Employee with Account

```
class Employee
{
  private String name;  // reference variable
  private double salary;
  private LocalDate hireDay; // reference variable
  private Account account;  // reference variable
  // Constructors
  public Employee(String n, double s, int year, int month,
     int day, Account a)
  {
     name = n;
     salary = s;
     hireDay = LocalDate.of(year, month, day);
     acct = a;
  }
 // Methods
 //  omitting getters and setters
```

```
public void raiseSalary(double byPercent)
   {
      double raise = salary * byPercent / 100;
      salary += raise;
   }

   public void payMonthly()
   {
       acct.deposit(salary/12.0);
   }
} // end of Employee
```

Data Classes

- 데이터 위주의 객체 정의 – depends on applications
  - Card (suit, value)
  - User authentication information (name, password)
  - Point (x, y)
- 객체 사이에 데이터 전달을 위해 사용
  - 매개변수 및 return value로 다중 데이터 전달
  - 객체 저장 및 읽기
- 구현 방법
  - public data
    - 코딩이 간결함
    - Read-only 객체를 구현하지 못함
    - Data validation check을 클래스 내부에서 구현하지 못함
    - Encapsulation 원칙 위배 (데이터 구조의 변화가 없을 때 사용)
  - private data – get/set method 제공
    - 코딩이 번거로움
    - Get method만 제공함으로써 read-only 객체를 구현할 수 있음 (e.g. Card data)
    - Set method에서 data validation check 가능 (e.g. password 특수문자 구성)

Data Classes : Data-oriented Classes

```
public class Point
{
  public int x;
  public int y;
  public Point(int x, int y)
  {
    this.x = x;
    this.y = y;
  }
}
…
Point p = new Point(4,5);
p.x = 5;
int yVal = p.y;
```

```
public class Point
{
  private int x;
  private int y;
  public Point(int x, int y)
  {
    this.x = x;
    this.y = y;
  }
  int getX() { return x; }
  int getY() { return y; }
  int void setX(int x) { this.x = x; }
  int void setY(int y) { this.y = y; }
}
…
Point p = new Point(4,5);
p.setX(5);
int yVal = p.getY();
```

Data Classes : Returning Multiple Values

```
public class MinMax
{
   public int min;
   public int max;
   public MinMax(int min, int max)
   {
      this.min = min;
      this.max = max;
   }
}
```

```
…
MinMax mm;
nt[] a = new int[10];
…
mm = minMax(a);
…
MinMax minMax(int a[])
{
   int min = Integer.MAX;
   int max = Integer.MIN;
   for(int t : a)
   {
      if ( t < min ) min = t;
      if (t > max ) max = t;
   }
   MinMax mm = new MinMax(min, max);
   return mm;
}
```

Data Classes : Immutable Data Class

```
public class Card
{
   private int suit;
   private int value;
   public Card(int s, int v)
   {
      suit = s;
      value = v;
   }
   int getSuit() { return suit; }
   int getValue() { return value; }
   // no setter methods
}
```