VISSOFT

Bài 5
Access modifier, từ khoá static, getter/setter

Module: ADVANCED PROGRAMMING WITH JAVA

Mục tiêu

VISSOFT

- Phân biệt được biến kiểu dữ liệu nguyên thủy và biến tham chiếu
- Phân biệt được biến của lớp và biến của đối tượng
- Phân biệt được phương thức của lớp và phương thức của đối tượng
- Khai báo và sử dụng được các biến static
- Khai báo và sử dụng được các phương thức static
- Trình bày được cú pháp khai báo package
- Trình bày được các access modifier
- Triển khai được getter/setter
- Trình bày được Nested Class
- Khai báo và sử dụng được Nested Class

VISSOFT

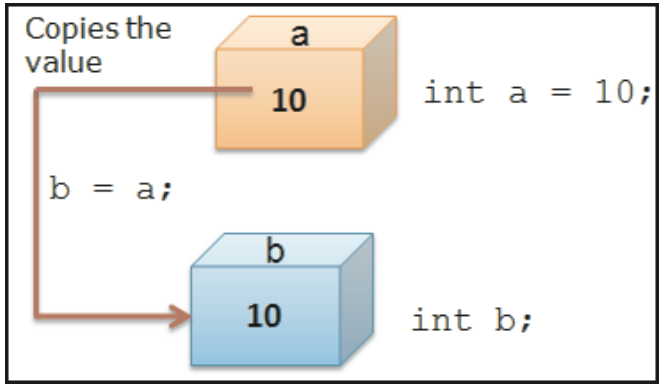
Biến tham chiếu và biến tham trị

Biến kiểu dữ liệu nguyên thuỷ Biến
thamchiếu

Primitive data types

- Các biến thuộc kiểu dữ liệu nguyên thuỷ (như int, long, float…) lưu trữ **giá trị** của chúng trong vùng nhớ được cấp
- Giá trị của một biến có thể được gán cho một biến khác
- Ví dụ:

```
int a = 10;
int b = a;
```



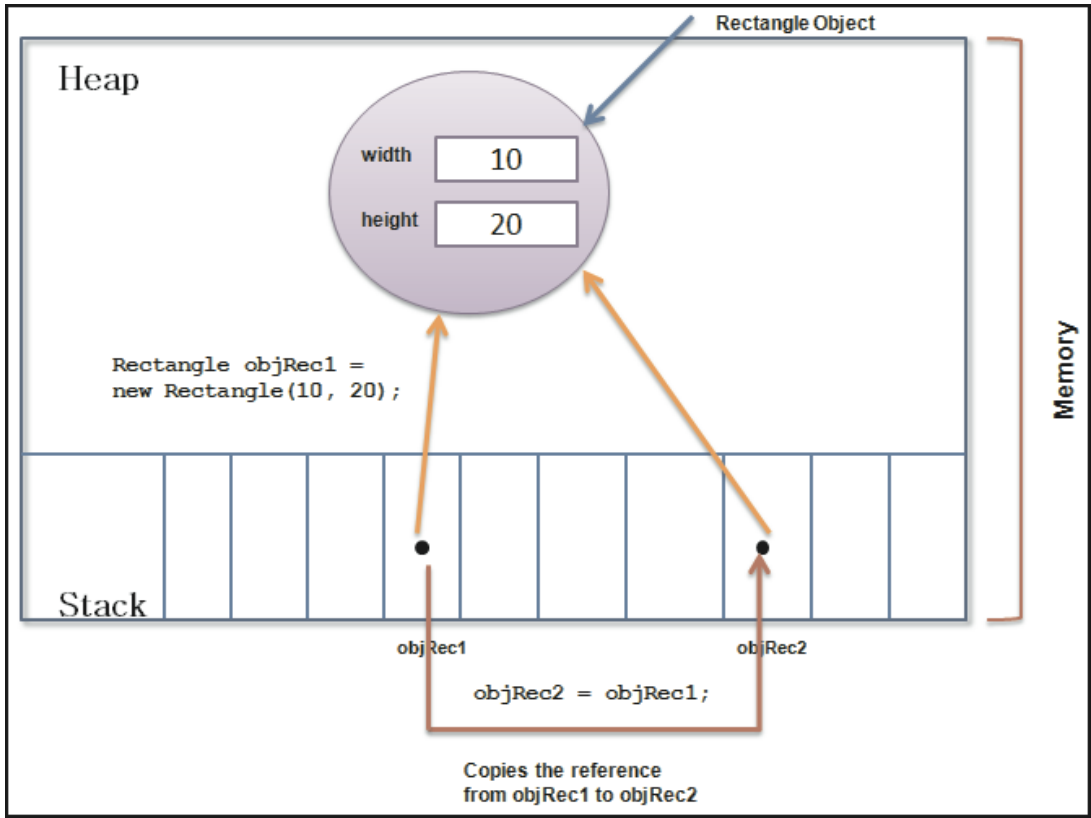
  - Thao tác này sao chép giá trị của biến a (được lưu trong vùng nhớ được cấp cho a) cho biến b (lưu vào vùng nhớ được cấp cho b)

Reference data types

- Các biến thuộc kiểu dữ liệu tham chiếu (chẳng hạn như Scanner, Person, Customer…) lưu trữ **tham chiếu** của đối tượng ở trong vùng nhớ được cấp
- Có thể gán giá trị tham chiếu của một biến cho một biến khác
- Ví dụ:

```
Rectangle rectangleObj1 = new Rectangle(10, 20);
Rectangle rectangleObj2 = rectangleObj1;
```

  - Thao tác này sao chép **địa chỉ** được lưu trong biến rectangleObj1 sang biến rectangleObj2
  - Không có ảnh hưởng nào xảy ra đối với đối tượng thực tế trong bộ nhớ



Primitive data type: Ví dụ

VISSOFT

```
public static void swap(int first, int second){
    int temp = first;
    first = second;
    second = temp;
}

public static void main(String[] args) {
    int a = 5;
    int b = 10;

    swap(a, b);

    System.out.println("a = " + a);
    System.out.println("b = " + b);
}
```

Kết quả:

```
a = 5
b = 10
```

Reference data type: Ví dụ

```
class Person{
    public String name;

    public Person(String name){
        this.name = name;
    }
}
```

```
public static void swap(Person first, Person second){
    String temp = first.name;
    first.name = second.name;
    second.name = temp;
}

public static void main(String[] args) {
    Person a = new Person("John");
    Person b = new Person("Bill");

    swap(a, b);

    System.out.println("a.name = " + a.name);
    System.out.println("b.name = " + b.name);
}
```

Kết quả:

```
a.name = Bill
b.name = John
```

VISSOFT

Từ khoá static

Static property

Static method

Từ khoá static

- Từ khoá *static* được sử dụng để khai báo các thuộc tính và phương thức của lớp (khác với thuộc tính và phương thức của đối tượng)
- Các thành phần static trực thuộc lớp, thay vì trực thuộc đối tượng
- Biến static còn được gọi là biến của lớp (class variable)
- Phương thức static còn được gọi là phương thức của lớp (class method)
- Có thể truy xuất các thành phần static bằng cách sử dụng lớp hoặc đối tượng
- Không cần khởi tạo đối tượng vẫn có thể sử dụng các thành phần static

Static property

- Cú pháp khai báo *static property*:

```
modifier static data_type variable_name;
```

- Ví dụ:

Khai báo biến static:

```
class Application{
    public static String language = "english";
}
```

Truy xuất biến static:

```
System.out.println("Current language: " + Application.language);
```

Static method

- Cú pháp khai báo static method:

```
modifier static data_type method_name(){
   //body
}
```

- Ví dụ:
  - Khai báo phương thức static

```
class Application{
   public static String getVersion(){
      return "1.0";
   }
}
```

  - Gọi phương thứcstatic

```
System.out.println("Current version: " + Application.getVersion());
```

Một số ràng buộc

- Phương thức static chỉ có thể gọi các phương thức static khác
- Phương thức static chỉ có thể truy xuất các biến static
- Phương thức static không thể sử dụng từ khoá *this* hoặc *super*
- Có thể khởi tạo biến static thông qua khối khởi tạo static
- Ví dụ:

```
class Application{
        public static String language;

        static {
                if(System.getProperty("lang").equals("en")){
                        language = "english";
                }else {
                        language = "spanish";
                }
        }
}
```

VISSOFT

Package

Package

- Package (gói) là cách để phân loại các lớp và interface thành các nhóm có liên quan đến nhau và tổ chức chúng thành các đơn vị để quản lý
- Ví dụ, Java cung cấp sẵn các gói:
  - *java.io*: Thực hiện các thao tác nhập xuất dữ liệu
  - *java.net*: Thực hiện các thao tác qua mạng lưới
  - *java.security*: Thực hiện các thao tác liên quan đến bảo mật
  - *java.util*: Cung cấp các lớp và phương thức hỗ trợ
  - ...
- Lập trình viên có thể tự định nghĩa các gói mới để tổ chức mã nguồn hợp lý

Tính chất của package

- Có thể khai báo các gói con -subpackage (gói ở bên trong gói)
- Không thể có 2 lớp có cùng tên trong cùng 1 gói
- Khi một lớp được khai báo bên trong một gói thì cần phải sử dụng tên của gói nếu muốn truy cập đến lớp đó
- Tên của gói được viết bằng chữ thường
- Các gói được cung cấp sẵn của Java được bắt đầu bằng từ *java* hoặc *javax*

Khai báo package

- Cú pháp:

```
package package_name;
```

- Ví dụ:

```
package codegym;
```

- Tên của package phải trùng với tên của thư mục chứa mã nguồn
- Tên của subpackage phải lần lượt trùng với tên của các thư mục tương ứng, ví dụ:

```
package com.codegym.ui;
```

Từ khoá import

- Cần sử dụng từ khoá import để có thể sử dụng các lớp được định nghĩa trong các package khác
- Ví dụ:

```
package model;

public class Customer {
}
```

```
package controller;

import model.Customer;

public class CustomerController {
    public void index(){
        Customer customer = new Customer();
    }
}
```

VISSOFT

Accessmodifier

Access modifier

VISSOFT

- Access modifier là các từ khoá được sử dụng để quy định mức độ truy cập đến lớp và các thành phần của lớp
- Các mức truy cập:
  - *public*: có thể truy cập từ bất cứ đâu
  - *private*: các phương thức và thuộc tính chỉ được phép truy xuất trong cùng một lớp
  - *protected*: các phương thức và thuộc tính được phép truy xuất trong cùng một lớp và ở các lớp con (kế thừa)
  - *default*: Nếu không có access modifier thì mức default sẽ được áp dụng. Lớp và các thành phần của lớp được truy xuất ở những nơi trong cùng một package

Access modifier: Ví dụ

VISSOFT

```java
package p1;

public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

```java
package p1;

public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;

    can invoke o.m1();
    can invoke o.m2();
    cannot invoke o.m3();
  }
}
```

```java
package p2;

public class C3 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    cannot access o.y;
    cannot access o.z;

    can invoke o.m1();
    cannot invoke o.m2();
    cannot invoke o.m3();
  }
}
```

Tổng hợp các mức truy cập

VISSOFT

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

VISSOFT

Getter và Setter

Truy cập trực tiếp vào các trường dữ liệu

VISSOFT

- Sử dụng từ khoá public khi khai báo thuộc tính sẽ cho phép truy cập trực tiếp vào các thuộc tính đó
- Ví dụ:

  Khai báo lớp Person sau cho phép truy cập trực tiếp vào trường name

```
class Person{
    public String name;
}
```

```
Person person = new Person();
person.name = "John";
```

- Nhược điểm:
  - Không kiểm soát được truy cập vào thuộc tính
  - Gây khó khăn cho việc duy trì, dễ phát sinh bug

Data field encapsulation

VISSOFT

- Data field encapsulation (bao gói trường dữ liệu) là hình thức hạn chế quyền truy cập trực tiếp vào các thuộc tính của đối tượng bằng cách sử dụng từ khoá private
- Khai báo các phương thức để kiểm soát việc truy cập vào các thuộc tính của đối tượng
- Các phương thức cho phép thay đổi giá trị của thuộc tính được gọi là setter, các phương thức cho phép lấy về giá trị của thuộc tính được gọi là getter
- Ví dụ getter: getName(), getAge(), getDate(), isAvailable()…
- Ví dụ setter: setName(), setAge(), setAddress()…

Khai báo getter/setter

- Cú pháp khai báo getter:

```
public returnType getPropertyName()
```

  - *Đối với các thuộc tính kiểu boolean thì tên getter bắt đầu bằng chữ is:*

```
public boolean isPropertyName()
```

- Cú pháp khai báo setter:

```
public void setPropertyName(dataType propertyValue)
```

Getter/setter: Ví dụ

VISSOFT

```
class Person{
    private String name;

    public void setName(String name){
        this.name = name;
    }

    public String getName(){
        return this.name;
    }
}
```

```
public static void main(String[] args) {
    Person person = new Person();
    person.setName("John");
    System.out.println("My name is: " + person.getName());
}
```

Từ khoá this

- Từ khoá *this* được sử dụng để đại diện cho đối tượng hiện tại
- Có thể sử dụng từ khoá this để truy cập đến các thành phần của đối tượng hiện tại
- Ví dụ, sử dụng từ khoá *this* để phân biệt 2 biến có cùng tên:

```
class Person{
    private String name;

    public void setName(String name){
        this.name = name;
    }
}
```

Biến name của lớp Person

Tham số name được truyền vào

VISSOFT

Nested class và anonymous class

Nested class

Anonymous class

Nestedclass

- Nested class (lớp lồng nhau) là một lớp *được* khai báo bên trong lớp khác
- Cú pháp:

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

Khi nào sử dụng nested class?

VISSOFT

- Để khai báo các lớp mà chỉ được sử dụng ở một nơi duy nhất
  - Ẩn một lớp ở bên trong lớp khác sẽ giúp cho các package gọn gàng hơn
- Để tăng tính bao gói (encapsulation)
- Giúp mã nguồn dễ đọc hơn
  - Các nested class nhỏ được đặt cạnh nơi sử dụng chúng giúp cho việc quản lý dễ dàng hơn

Static nested class

- Static nested class trực thuộc lớp ở bên ngoài (thay vì trực thuộc đối tượng của lớp bên ngoài)
- Static nested class không thể truy xuất đến các thành phần của lớp bên ngoài
- Sử dụng tên của lớp bên ngoài để truy cập đến lớp bên trong
- Ví dụ:

```
OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();
```

Inner class (non-static class)

VISSOFT

- Inner class trực thuộc một đối tượng của lớp bên ngoài
- Inner class có thể truy xuất đến các thành phần của lớp bên ngoài
- Sử dụng tham chiếu của một đối tượng của lớp bên ngoài để truy xuất đến Inner class
- Ví dụ:

```
OuterClass outerObject = new OuterClass();
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

Local class

- Local class (lớp địa phương) là lớp được khai báo và sử dụng bên trong một khối lệnh
- Không thể sử dụng local class bên ngoài khối lệnh được khai báo
- Ví dụ:

```
class Customer{
    public boolean validateAddress(String customerAddress){
        class Address{
            String address;
            Address(String address){
                this.address = address;
            }

            public boolean validate(){
                //body
            }
        }

        Address address = new Address(customerAddress);
        return address.validate();

    }
}
```

VISSOFT

Demo

Tổng kết

VISSOFT

- Biến tham trị chứa giá trị của nó trong vùng nhớ được cấp
- Biến tham chiếu chứa tham chiếu đến đối tượng trong vùng nhớ được cấp
- Từ khoá static được sử dụng để khai báo các thành phần thuộc lớp
- package được sử dụng để nhóm các lớp có liên quan đến nhau trong cùng một đơn vị
- Getter/setter là cơ chế để kiểm soát truy cập đến các trường dữ liệu của đối tượng
- Nested class là lớp được khai báo bên trong lớp khác
- Local class là lớp được khai báo bên trong một khối lệnh