

BÀI 2. CƠ BẢN VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

Các khái niệm cơ bản của lập trình hướng đối tượng
Khai báo và sử dụng class trong Java

1

1

1. CÁC KHÁI NIỆM CƠ BẢN

2

2

Lập trình hướng đối tượng là gì?

- Mô hình hóa các đối tượng trong thế giới thực thành đối tượng phần mềm
- **Chương trình = Đối tượng + Thông điệp**
- Chương trình được cấu thành bởi các đối tượng và **tương tác giữa các đối tượng (qua thông điệp)**
- Thuộc tính: các đặc điểm, trạng thái của đối tượng
- Phương thức: các hành vi của đối tượng

3

3

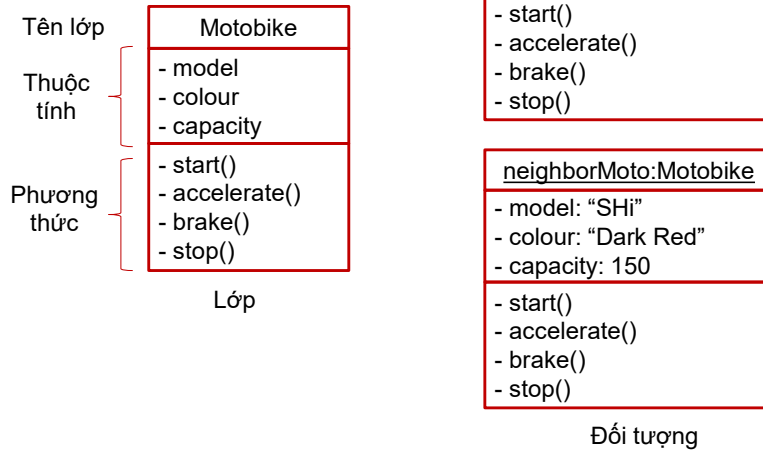
Lớp vs đối tượng

- **Lớp (Class)**: định nghĩa các thuộc tính và các phương thức **chung** của một nhóm đối tượng nào đó
 - Lớp là **trừu tượng**, thuộc tính không mang giá trị cụ thể
 - Có thể liên tưởng đến kiểu dữ liệu
- **Đối tượng (Object)**: là một thể hiện **cụ thể** của lớp, các thuộc tính có giá trị xác định
 - Có thể liên tưởng đến biến
- Lớp là mô hình hóa rút gọn của thực thể trên thực tế
 - Chỉ mô tả những thuộc tính, phương thức quan tâm

4

4

Lớp vs đối tượng



5

5

Các nguyên lý cơ bản của HĐT



6

6

Trừu tượng hóa

- Loại bỏ đi các thông tin cụ thể, giữ lại các thông tin chung
- Tập trung vào các đặc điểm của thực thể, làm cho nó khác biệt với những thực thể khác
- Phụ thuộc góc nhìn

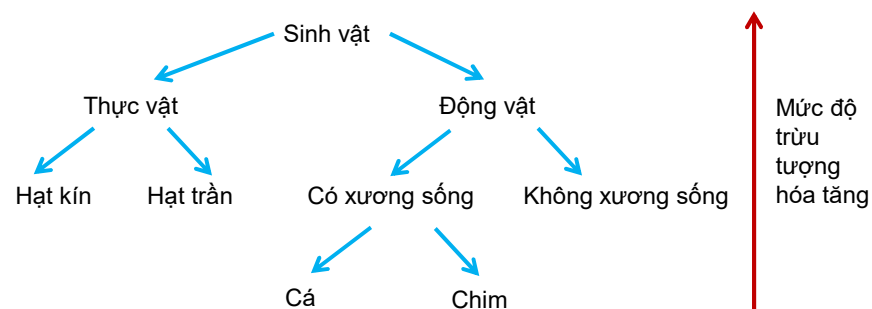
7

7

Phân cấp

- Một nhóm đối tượng mang những đặc điểm khác biệt với những đối tượng khác có thể tách thành nhóm con

→ Lặp lại bước trên ta có cây phân cấp

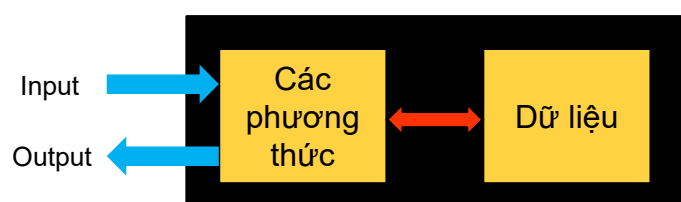


8

8

Mô-đun hóa và Đóng gói

- Mô-đun hóa: Chia nhỏ hệ thống phức tạp thành các đối tượng nhỏ hơn
- Đóng gói: Che giấu, ẩn chi tiết thực hiện bên trong một đối tượng
 - Một đối tượng có 2 khung nhìn: từ bên trong, từ bên ngoài
 - Cung cấp cho các đối tượng khác (client) một giao diện
 - Tính trong suốt: Thay đổi việc thực thi bên trong không làm ảnh hưởng tới các đối tượng khác



9

9

2. ĐÓNG GÓI VÀ XÂY DỰNG LỚP

10

10

Đóng gói

- Lớp đóng gói các thành viên và chỉ định điều khiển truy cập tới các thành viên đó:
 - Thuộc tính
 - Phương thức
- Tập hợp các lớp được nhóm lại thành gói (package). Mỗi lớp trong gói cũng được chỉ định điều khiển truy cập
- Các từ khóa chỉ định điều khiển truy cập trong Java:
 - **public**: có thể truy cập **từ mọi nơi**
 - **protected**: có thể truy cập từ trong gói hoặc từ các lớp con
 - Không chỉ định: có thể truy cập từ trong gói
 - **private**: chỉ có thể truy cập từ **chính lớp đó**

11

11

Mô tả phần tiêu đề của lớp

- Cú pháp

```
package pack.name;
/**Javadoc*/
modifier class ClassName{
    //class's body
}
```

} Tiêu đề của lớp
(class header)

- Trong đó:

pack.name : tên gói

/**Javadoc*/: chú thích để tạo tài liệu Javadoc (không bắt buộc)

modifier : **chỉ định truy cập** (chỉ có thể là **public** hoặc không chỉ định)

ClassName : tên lớp, theo quy tắc định danh của Java

12

12

Khai báo thuộc tính

- Cú pháp

```
/**Javadoc*/
modifier DataType attributeName;
```

- Trong đó:

/**Javadoc*/: chú thích để tạo tài liệu Javadoc (không bắt buộc)

modifier : chỉ định truy cập **public**(truy cập được từ mọi nơi) hoặc không chỉ định

Datatype : kiểu dữ liệu, có thể là một lớp lồng (nested class)

attributeName : tên thuộc tính, theo quy tắc định danh

13

13

Khai báo thuộc tính hằng số

- Cú pháp:

```
modifier final DataType CONST_NAME = value;
```

- Trong đó:

modifier : chỉ định truy cập

Datatype : kiểu dữ liệu

CONST_NAME : tên hằng

value : giá trị gán cho hằng

- Ví dụ

```
public final int MAX_STUDENT = 100;
```

14

14

Khai báo phương thức

- Cú pháp

```
/**Javadoc*/
modifier DataType methodName (parameterList) {
    // method's body
}
```

- Trong đó:

`/**Javadoc*/`: chú thích để tạo tài liệu Javadoc (không bắt buộc)

`modifier` : chỉ định truy cập

`DataType` : kiểu dữ liệu trả về của phương thức

`methodName` : tên phương thức

`parameterList` : danh sách các tham số, bao gồm kiểu dữ liệu và tên của tham số

15

15

Lệnh return

- Nếu `DataType` là `void`: phương thức không trả về giá trị
- Ngược lại, trong nội dung của phương thức cần sử dụng lệnh `return` để trả về giá trị có kiểu dữ liệu tương ứng
- Khi câu lệnh `return` được thực hiện, phương thức sẽ kết thúc
 - Cần cẩn trọng khi sử dụng lệnh `return` ở giữa phương thức vì nó sẽ làm thay đổi cấu trúc lập trình.

--> Trong một phương thức: nếu có `void` thì không cần `return` giá trị và ngược nếu không `void` thì cần phải `return` giá trị

16

16

Các phương pháp truyền tham số

- Khi xây dựng các phương thức có nhận tham số, Java luôn sử dụng cách thức truyền theo kiểu tham trị (pass-by-value), tức là **chỉ truyền vào bản sao/giá trị** của tham số:
 - Tham số có kiểu dữ liệu nguyên thủy: truyền vào giá trị của tham số. Do đó mọi thay đổi trên giá trị này của phương thức không ảnh hưởng tới bản gốc.
 - Tham số có kiểu dữ liệu tham chiếu: truyền bản sao của tham chiếu gốc
- Chúng ta sẽ phân tích kỹ hơn vấn đề này sau

17

17

Phương thức khởi tạo (constructor)

- Khi một **đối tượng tạo ra**, một phương thức đặc biệt được gọi là **phương thức khởi tạo** được tự động gọi:
 - Gán giá trị ban đầu cho các thuộc tính
 - Có thể thực hiện một số thao tác xử lý
 - Một lớp có thể có nhiều phương thức khởi tạo
- Cú pháp

```
/**Javadoc*/
Modifier ClassName(parameterList) {
    // method's body
}
```

- Phương thức khởi tạo mặc định (nên có)

```
/**Javadoc*/
Modifier ClassName() {
    // method's body
}
```

18

18

class: thuộc tính + phương thức chung của một nhóm đối tượng

- Đối tượng: cụ thể của class + thuộc tính có giá trị xác định

Thành viên static

- Thành viên static: các thành viên có thể được truy cập từ tên lớp mà không cần qua một đối tượng (thể hiện của lớp)
- Cách thức: khai báo thành viên với từ khóa `static`

```
modifier static final DataType CONST_NAME = value;
```

```
modifier static DataType attributeName;
```

```
modifier static DataType methodName(parameterList) {
    // method's body
}
```

- Một phương thức `static` chỉ có thể truy cập vào các thuộc tính `static` và chỉ có thể gọi các phương thức `static` cùng lớp

19

19

Thành viên vs Thành viên static

Thành viên

- Chỉ được truy cập thông qua một đối tượng (thể hiện của lớp)
- Thuộc tính của các đối tượng có giá trị khác nhau
- Sự thay đổi giá trị thuộc tính của một đối tượng này là độc lập với các đối tượng khác

Thành viên static

- Có thể truy cập thông qua tên lớp hoặc qua đối tượng
- Thuộc tính của đối tượng khác nhau có giá trị giống nhau
- Sự thay đổi giá trị thuộc tính của đối tượng này kéo theo sự thay đổi tương ứng trên các đối tượng khác

20

20

Thành viên lớp – Ví dụ 1

```

/** The StaticMethodTest class illustrates using static
method */
public class StaticMethodTest {
    /** The main method begins execution of Java
    application
    *@param args: input parameter
    */
    public static void main (String[] args){
        char[] inputStr = {'S','o','I','C','T'};
        String data = String.valueOf(inputStr);
    }
}

```

--> phương thức có void -> không cần return giá trị
 --> phương thức có static --> không cần truy cập vào đối tượng
 có thể truy cập vào class

21

21

Thành viên lớp – Ví dụ 2

```

package java.oop.example
/** The StaticAttribute class illustrates declaring static
attribute */
public class StaticAttribute {
    private static int staticAttr;
    /** Set a new value to staticAttr field
    *@param newValue: new value
    */
    public void setStaticAttr (int initValue){
        staticAttr = initValue;
    }
    /** Gets the value of staticAttr*/
    public int getStaticAttr (){
        return staticAttr;
    }
}

```

22

22

Thành viên lớp – Ví dụ 2

```
package java.oop.example.static
import java.oop.example.StaticAttribute
/** The StaticAttributeTest class illustrates accessing
static attribute */
public class StaticAttributeTest {

    public static void main (String[] args) {
        StaticAttribute attr1 = new StaticAttribute();
        attr1.setStaticAttr(1);
        StaticAttribute attr2 = new StaticAttribute ();
        attr2.setStaticAttr(-1);
        System.out.println("Static attribute in attr1: " +
            attr1.getStaticAttr());
        System.out.println("Static attribute in attr2: " +
            attr2.getStaticAttr());
    }
}
```

23

23

Bảo vệ dữ liệu của đối tượng

- Thuộc tính của lớp thường được đặt **chỉ định truy cập** là **private** (**modifier**)
- Truy cập vào **thuộc tính** private thông qua **các hàm setter** (thay đổi giá trị của thuộc tính) và **hàm getter** (lấy giá trị của thuộc tính)
- Mục đích: **che giấu và kiểm soát giá trị truyền cho thuộc tính**

```
private int intAttr;

//Setter method
public void setIntAttr (int intAttr){
    this.intAttr = intAttr;
}
//Getter method
public int getIntAttr (){
    return this.intAttr;
}
```

Từ khóa **this** cho phép **tự tham chiếu đến chính đối tượng đó**

24

24

Ví dụ về xây dựng lớp

- Tạo và thử một đối tượng “học phần” có
 - Mã môn, tên môn, số lượng đăng ký tối đa và số lượng đăng ký hiện tại
 - Cho phép người dùng đăng ký một SV vào lớp, hủy đăng ký một SV và hiển thị thông tin môn học
- Yêu cầu :
 - Không thể đăng ký nữa nếu số đăng ký đã bằng định mức
 - Không thể hủy đăng ký nếu không có sinh viên

25

25

Lớp Subject

- Phát hiện thuộc tính:
 - `subjectID`: Mã học phần – Xâu ký tự
 - `subjectName`: Tên học phần – Xâu ký tự
 - `quota`: Số lượng sinh viên tối đa được đăng ký
 - `currentEnrolment`: Số SV đăng ký hiện tại
- Phát hiện phương thức
 - `enrolStudent()`: Đăng ký một sinh viên
 - `unEnrolStudent()`: Hủy đăng ký
 - `displaySubjectInfo()`: Hiển thị thông tin đăng ký
 - Các phương thức setter và getter
 - Phương thức khởi tạo

26

26

Định nghĩa lớp Subject

```
package java.oop.subject
/** The Subject class illustrates a subject in HUST */
public class Subject {
    private String subjectID;
    private String subjectName;
    private int quota;
    private int currentEnrolment;

    /** The default constructor
    public Subject(){
        this.quota = 0;
    }
    //Declare other methods...
}
```

chỉ định truy cập lớp

chỉ định truy cập thành viên

this cho phép sự tự tham chiếu đến chính đối tượng đó

27

27

phương thức khởi tạo --> tạo ra một đối tượng Constructor

```
/** Constructs a newly created Subject object by
    assigning initial values for attributes
    *@param initID : the ID of subject
    *@param initName : the name of subject
    *@param initQuota : the quota of subject
    *@param initCurrentEnrolment : the initial number of
        students enrolling
    */
public Subject(String initID, String initName, int
    initQuota, int initCurrentEnrolment){
    this.subjectID = new String(initID);
    this.subjectName = new String (initName);
    this.quota = initQuota;
    this.currentEnrolment = initCurrentEnrolment;
}
```

chỉ định truy cập thành viên

28

28

enrolStudent()

```
/** Enrols a student into course*/

public void enrolStudent(){
    System.out.print("Enrolling student... ");
    if (currentEnrolment < quota){
        ++currentEnrolment;
        System.out.println("Student enrolled in " +
                           subjectName);
    }
    else
        System.out.println("Quota reached, enrolment
                           failed");
}
```

29

29

unEnrolStudent()

```
/** Cancels an enrolment*/

public void unEnrolStudent(){
    System.out.print("Un-enrolling student... ");
    if (currentEnrolment <= 0)
        System.out.println("No students to un-enrol");
    else{
        --currentEnrolment;
        System.out.println("Student un-enrolled from " +
                           subjectName);
    }
}
```

30

30

displaySubjectInfo()

```
/** Displays the information of the course*/

public void displaySubjectInfo(){
    System.out.println("Subject ID: " + subjectID);
    System.out.println("Subject name: " + subjectName);
    System.out.println("Quota: " + quota);
    System.out.println("Currently enrolled: " +
                       currentEnrolment);

    int availablePlaces = quota - currentEnrolment;
    System.out.println("Can accept " + availablePlaces +
                       " more students");
}
```

31

31

3. KHAI BÁO VÀ SỬ DỤNG ĐỐI TƯỢNG

32

32

Khai báo và sử dụng đối tượng

```
package java.oop.example.static
import java.oop.example.StaticAttribute
/** The StaticAttributeTest class illustrates accessing
static attribute */
public class StaticAttributeTest {

    public static void main (String[] args) {
        StaticAttribute attr1 = new StaticAttribute ();
        attr1.setStaticAttr(1);
        StaticAttribute attr2 = new StaticAttribute ();
        attr1.setStaticAttr(-1);
        System.out.println("Static attribute in attr1: " +
            attr1.getStaticAttr());
        System.out.println("Static attribute in attr2: " +
            attr2.getStaticAttr());
    }
}
```

33

33

Khai báo và sử dụng đối tượng

- Cú pháp

```
ClassName objectName = new ClassName();
```

- Trong đó:

ClassName : Tên lớp mà đối tượng tạo ra từ lớp đó

objectName : Tên đối tượng

ClassName() : Phương thức khởi tạo. Tên phương thức khởi tạo trùng với tên lớp.

- Truy cập tới các thành viên của đối tượng qua toán tử '.'
- Lưu ý tới chỉ định điều khiển truy cập
- Các đối tượng được khai báo mà không khởi tạo sẽ mang giá trị null

34

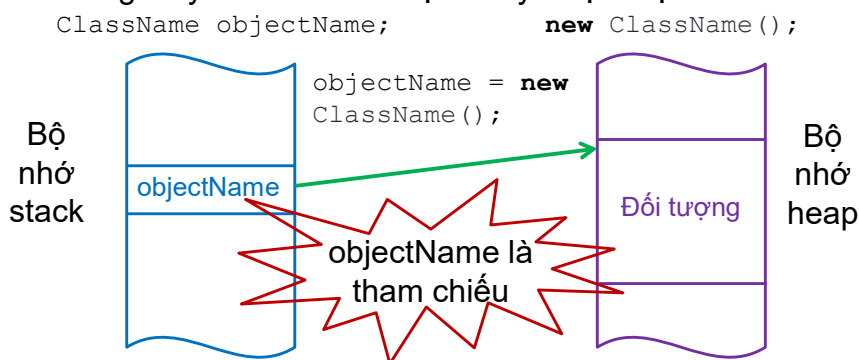
34

Khai báo và sử dụng đối tượng (tiếp)

- Có thể tách thành 2 câu lệnh:

```
ClassName objectName;  
objectName = new ClassName();
```

- Điều gì xảy ra khi hai câu lệnh này được thực thi:



Giá trị của objectName là địa chỉ vùng nhớ chứa thông tin của đối tượng trong bộ nhớ heap

35

35

Tạo và sử dụng đối tượng Subject

```
package java.oop.subject;  
/** The SubjectManagement class manages subjects*/  
public class SubjectManagement {  
    public static void main(String[] args) {  
        Subject javaPrograming = new Subject("IT3650",  
                                                "Java Programing", 40,0);  
        javaPrograming.unEnrolStudent();  
        javaPrograming.displayInfor();  
  
        for (int i = 1; i <= 40; i++)  
            javaPrograming.enrolStudent();  
        javaPrograming.displayInfor();  
  
        javaPrograming.enrolStudent();  
        javaPrograming.displayInfor();  
    }  
}
```

36

36

Truyền tham số cho phương thức

- Nhắc lại: Java sử dụng các thức truyền theo tham trị (pass-by-value)
 - Khi truyền tham số có kiểu tham chiếu, một bản sao sẽ được tạo ra:
 - Giá trị tham chiếu gốc không bị thay đổi sau khi kết thúc phương thức
 - Giá trị của đối tượng giữ lại mọi thay đổi mà phương thức đã tạo ra
- hiệu quả tương tự như truyền tham biến

37

37

Ví dụ - Truyền tham số cho phương thức

```
package java.oop.example;
/** The AnyValue class contains a integer */
public class AnyValue {
    private int data;

    /** Construcs a new object with initial value*/
    public AnyValue (int initData){
        this.data = initData;
    }
    /** Setter method*/
    public void setData (int newData){
        this.data = newData;
    }
    /** Getter method*/
    public int getData(){
        return this.data;
    }
}
```

38

38

Truyền tham số cho phương thức

```
package java.oop.example.passing;
import java.oop.example.AnyValue
/**This class illustrates passing parameter when calling
a method */
public class PassingParameter {
    /** Method has a primitive value*/
    private static void changeNumber(int a){
        a = a + 1;
    }
    /** Method has a reference value*/
    private static void changeValue(AnyValue value){
        value.setData(10);
    }
}
```

39

39

Truyền tham số cho phương thức (tiếp)

```
public static void main(String[] args){
    //pass primitive parameter
    int number = 1;
    System.out.println("Before changing: number = "
        + number);
    changeNumber(number);
    System.out.println("After changing: number = "
        + number);
    //pass reference parameter
    AnyValue value = new AnyValue(1);
    System.out.println("Before changing: value has " +
        value.getData());
    changeValue(value);
    System.out.println("After changing: value has " +
        value.getData());
}
}
```

40

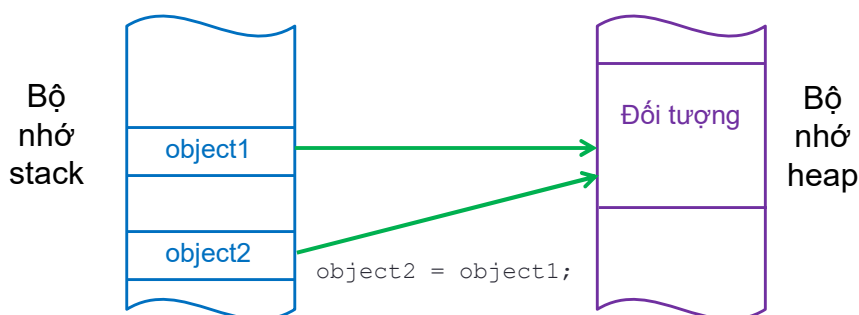
40

Phép gán đối tượng

- Java cho phép gán hai đối tượng cho nhau:

```
object2 = object1;
```

- Hãy nhớ rằng `object1` và `object2` chỉ là các tham chiếu. Do đó câu lệnh gán trên chỉ là gán giá trị tham chiếu. Hay nói cách khác, một thể hiện sẽ có hai tham chiếu `object1` và `object2`



41

41

So sánh hai đối tượng

- So sánh hai giá trị có kiểu nguyên thủy: `==`
- So sánh hai đối tượng : `object1 == object2` ?
 - Luôn nhớ rằng `object1` và `object2` chỉ là tham chiếu. Vì vậy toán tử `==` chỉ so sánh giá trị của 2 tham chiếu
 - Kiểm tra 2 tham chiếu có cùng trỏ đến một đối tượng không
- Phương thức `equals (Object)`
 - Các lớp của Java:
 - Trả về `true` nếu mọi thành viên của 2 đối tượng có giá trị bằng nhau
 - Trả về `false` nếu ngược lại
 - Các lớp định nghĩa mới: Cần sử dụng nguyên lý chồng phương thức để viết lại. Chúng ta sẽ đề cập đến sau.

42

42

Ví dụ - Gán và so sánh hai đối tượng

```
package java.oop.example.comparison;
import java.oop.example.AnyValue
/** The ObjectComparision class illutrates comparing two
object*/
public class ObjectComparison {
    public static void main(String[] args) {
        AnyValue value1 = new AnyValue(1);
        AnyValue value2 = new AnyValue(1);
        System.out.println("***Compare two object:");
        System.out.println("value1 == value2: " +
            (value1==value2));
        System.out.println("value1.equals(value2): " +
            value1.equals(value2));
    }
}
```

43

43

Ví dụ - Gán và so sánh hai đối tượng (tiếp)

```
        System.out.println("***After assigning value2 to
                                value1:");
        value1 = value2;
        System.out.println("value1 == value2: " +
            (value1==value2));
        value1.setData(100);
        System.out.println("The data of value1: " +
            value1.getData());
        System.out.println("The data of value2: " +
            value2.getData());
    }
}
```

44

44

Mảng đối tượng

- Cú pháp khai báo và khởi tạo

```
ClassName[] arrayName = new ClassName[size];
```

- Trong đó:

ClassName : Tên lớp mà đối tượng tạo ra từ lớp đó

arrayName : Tên mảng đối tượng

size : Kích thước

- Các phần tử sẽ mang giá trị null

45

45

Mảng đối tượng

- Cú pháp

```
ClassName[] arrayName = {new ClassName(), ...,  
                          new ClassName()};
```

- Trong đó:

ClassName : Tên lớp mà đối tượng tạo ra từ lớp đó

arrayName : Tên mảng đối tượng

- Số lần gọi phương thức khởi tạo là kích thước của mảng

46

46

Bài tập

- Viết thêm các phương thức setter và getter cho lớp `Subject`
- Viết lại lớp `SubjectManagement` cho phép thực hiện những công việc sau:
 - Nhập thông tin môn học từ bàn phím
 - Tạo một môn học mới theo thông tin người dùng nhập từ bàn phím
 - Cho phép đăng ký thêm một sinh viên. Mỗi lần đăng ký xong, hỏi lại người dùng có tiếp tục không. Hiển thị thông tin môn học nếu người dùng không muốn tiếp tục.
 - Cho phép hủy đăng ký một sinh viên. Mỗi lần hủy đăng ký xong, hỏi lại người dùng có tiếp tục không. Hiển thị thông tin môn học nếu người dùng không muốn tiếp tục.

47

47

3. MỘT SỐ LỚP TIỆN ÍCH TRONG JAVA

48

48

3.1. Các lớp bao (Wrapped class)

- Các kiểu dữ liệu nguyên thủy không có các phương thức kèm theo
- Java xây dựng lớp bao tương ứng cho các kiểu dữ liệu nguyên thủy:
 - Mỗi đối tượng của lớp bao sẽ chứa giá trị và các phương thức để xử lý
- Các lớp bao trong Java

Lớp bao	Kiểu nguyên thủy
Boolean	boolean
Byte	byte
Char	char
Short	short
Integer	int
Long	long
Float	float
Double	double

49

49

Một số phương thức điển hình

- `String toString()`: Chuyển giá trị thành chuỗi
- `<type>Value`: Chuyển từ đối tượng lớp bao thành giá trị nguyên thủy tương ứng

```
Float fObject = 3.14;
float fValue = fObject.floatValue(); //fValue = 3.14f
int iValue = fObject.intValue(); //iValue = 3
```

- `int compareTo(Wrapped)`: so sánh giá trị với đối tượng khác cùng lớp bao

50

50

Một số phương thức static

- `parse<type>(String)`: chuyển chuỗi thành giá trị tương ứng
- `toString(value)`: chuyển giá trị thành chuỗi tương ứng
- `max(value1, value2)`: giá trị lớn hơn trong hai giá trị
- `min(value1, value2)`: giá trị nhỏ hơn trong hai giá trị

```
float fValue = Float.parseFloat("3.14"); //fValue = 3.14f
String str = Float.toString(fValue); //str = "3.14"
```

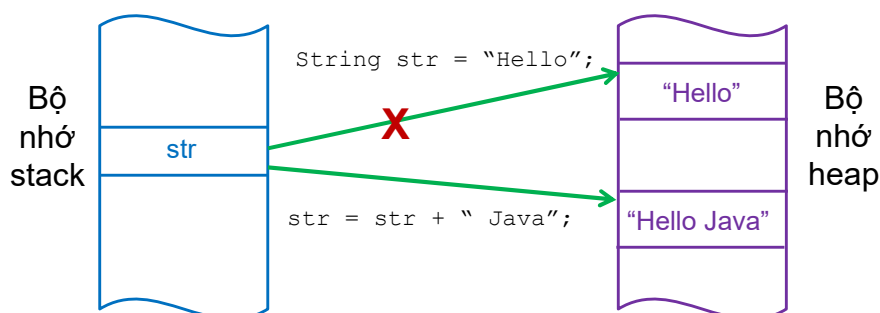
51

51

3.2. Lớp StringBuffer và StringBuilder

- Đối tượng `String` không thay đổi được giá trị. Xét ví dụ:

```
String str = "Hello";
str = str + " Java"; //str = "Hello Java"
```



Vùng nhớ chứa nội dung cũ không bị thu hồi ngay → vùng nhớ rác

52

52

Lớp StringBuffer và StringBuilder

- Lớp StringBuffer và StringBuilder cho phép làm việc với xâu linh hoạt hơn, hiệu quả hơn về mặt cấp phát tài nguyên
- Dùng StringBuffer khi có nhiều tiến trình cùng truy cập tới một xâu
- Dùng StringBuilder khi có 1 tiến trình truy cập tới xâu
 - Trường hợp này nếu dùng StringBuffer sẽ cho hiệu năng kém hơn
- Các phương thức của 2 lớp giống nhau

53

53

Các phương thức khởi tạo

- StringBuffer(): Khởi tạo 1 đối tượng với kích thước ban đầu 16 ký tự, không chứa ký tự nào
- StringBuffer(String str): Khởi tạo đối tượng với nội dung ban đầu là str
- StringBuffer(char ch): Khởi tạo 1 đối tượng với nội dung là ký tự ch
- StringBuffer(int size): Khởi tạo đối tượng với kích thước ban đầu là size
 - Lưu ý: khi xử lý đối tượng, kích thước có thể thay đổi tùy ý
- Tương tự với StringBuilder

54

54

Một số phương thức điển hình

- `toString()`: trả lại xâu nội dung có trong đối tượng `StringBuffer` hay `StringBuilder`
- `append(value)`: thêm giá trị (có kiểu bất kỳ) vào nội dung của đối tượng
- `insert(int position, value)`: chèn giá trị vào sau vị trí `position`
- `capacity()`: Kích thước vùng nhớ để chứa xâu nội dung
- `length()`: độ dài của xâu nội dung
- `charAt (int index)`: trả lại ký tự có chỉ số `index`
- `delete(int start, int end)`: xóa xâu con từ vị trí `start` đến vị trí `end`

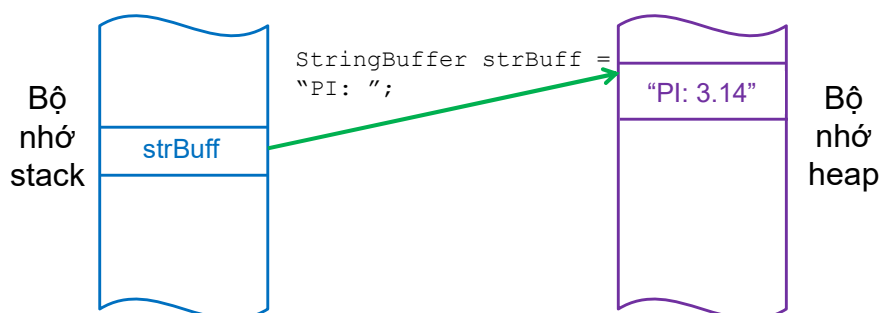
55

55

StringBuffer – Ví dụ

```
StringBuffer strBuff = "PI: ";
strBuff.append(3.14);
```

```
strBuff.append(3.14);
```



56

56