BIẾN, HẰNG VÀ CÁC KIỂU DỮ LIỆU

NỘI DUNG TRONG CHƯƠNG

- Khái niệm về hằng, biến
- Phân loại biến
- Các quy tắc đặt tên
- Các kiểu dữ liệu cơ bản
- Khái niệm toán tử và các loại toán tử

Chương 2 có nội dung liên quan đến các khái niệm cơ bản nhất trong Java bao gồm: hằng, biến, kiểu dữ liệu và toán tử. Java cũng đưa ra các quy tắc và quy ước cho việc đặt tên như đặt tên biến, tên lớp, tên phương thức, tên hằng số ... Việc tuân thủ các quy tắc đặt tên giúp tạo ra những khai báo hợp lệ, đồng thời giúp mã nguồn trở lên dễ đọc và bảo trì hơn. Chương này cũng đề cập đến cách sử dụng chuỗi, bao gồm cách khai báo và khởi tạo chuỗi ký tự và cách sử dụng các phương thức để làm việc với chuỗi. Phần cuối của chương là cách sử dụng năm loại toán tử phổ biến trong Java bao gồm toán tử số học, toán tử gán, toán tử so sánh, toán tử logic và toán tử điều kiện.

ĐẶT TÊN TRONG JAVA

Trong các chương trình, **đặt tên** (hay còn gọi là **định danh)** được dùng để xác định các thành phần khác nhau trong chương trình. Đó có thể là **biến, hàm, lớp, đối tượng, phương thức**...

Quy tắc đặt tên

Mỗi một ngôn ngữ lập trình khác nhau thường sẽ có **quy tắc** đặt tên khác nhau. Java cũng có những quy tắc đặt tên cần **phải** tuân theo, cụ thể như sau:

- Không được sử dụng các **từ khóa** để đặt tên (tra cứu từ khóa trong phần mục lục).
- Chỉ được phép sử dụng các **chữ cái**, **chữ số**, dấu **$** và dấu **_** khi đặt tên.
- Tên **không** được bắt đầu bởi chữ số.

Một số ví dụ về việc đặt tên hợp lệ:

- age
- $salary
- _value
- Student
- NUMBER1

Một số ví dụ về việc đặt tên không hợp lệ:

- 123abc
- public
- -salary
- @test

Cần lưu ý rằng, không chỉ Java, hầu hết các ngôn ngữ lập trình đều không cho phép đặt tên có chứa khoảng trống. Khi đó **HelloWorld** là một định danh hợp lệ, nhưng **Hello World** thì không. Java phân biệt chữ hoa và chữ thường do đó HelloWorld, helloworld, HELLOWORLD và hElloWorLD đều là những tên riêng biệt.

Quy ước đặt tên

Bên cạnh những quy tắc về việc đặt tên mã mỗi người viết mã nào cũng **phải** tuân theo. Việc đặt tên còn **nên** tuân theo một số **quy ước** như sau:

Quy ước đặt tên Lớp (Class):

- Nên bắt đầu bằng chữ viết hoa.
- Là một danh từ (như Color, Button, System, Topic)
- Sử dụng các từ thích hợp, thay vì các từ viết tắt

Ví dụ:

```
class Employee{
}
```

Quy ước đặt tên phương thức (Method):

- Bắt đầu bằng chữ thường.
- Là một động từ như main(), print(), println().
- Nếu tên chứa nhiều từ, hãy bắt đầu nó bằng một chữ cái viết thường theo sau là một chữ cái viết hoa như (camelCase) như actionPerformed().

Ví dụ:

```
class Employee{
  //method
  void draw(){
  }
}
```

Quy ước đặt tên Biến (Variable):

- Bắt đầu bằng một chữ cái viết thường như id, name

- Nếu tên chứa nhiều từ, hãy bắt đầu bằng chữ cái viết thường theo sau là chữ cái viết hoa như firstName, lastName,...
- Tránh sử dụng các biến một ký tự như x, y, z, hoặc các biến không miêu tả được ý nghĩa của biến.

Ví dụ khai báo một biến như một thuộc tính của lớp Employee:

```
class Employee{
  //variable
  int id;
}
```

Quy ước đặt tên Giao diện (Interface):

- Nên bắt đầu bằng chữ viết hoa (thường 1 chữ I)
- Theo sau là một tính từ như Runnable, Remote, ActionListener
- Sử dụng các từ thích hợp, thay vì các từ viết tắt

Ví dụ:

```
interface IPrintable{
}
```

Quy ước đặt tên Gói (Package):

- Tên nên là một chữ cái viết thường như java, lang.
- Nếu tên chứa nhiều từ, nó nên được phân tách bằng dấu chấm (.) Như java.util, java.lang.

Ví dụ:

```
package com.javatpoint; //package
class Employee{
}
```

Quy ước đặt tên Hằng (Constant):

- Nên được viết bằng chữ in hoa như RED, YELLOW.
- Nếu tên chứa nhiều từ, nó nên phân tách bằng dấu gạch dưới (_), chẳng hạn như MAX_PRIORITY.

Ví dụ:

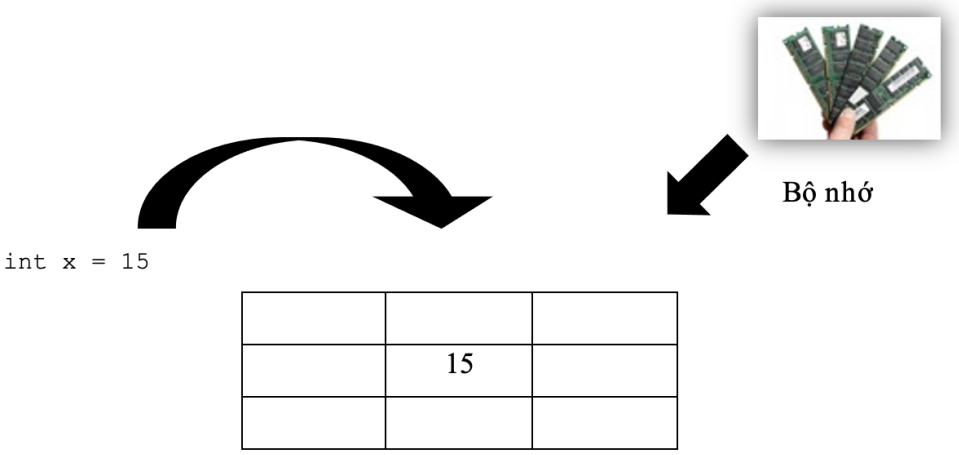
```
class Employee{
  //constant
  static final int MIN_AGE = 18;
}
```

BIẾN

Biến là gì

Biến (variable) là giá trị **có thể thay đổi** được trong mỗi chương trình. Một biến sẽ được cấp phát một **vùng nhớ** trong bộ nhớ chính để lưu trữ giá trị. Mỗi biến phải được đặt một tên duy nhất (định danh). Để khai báo biến cần khai báo kiểu dữ liệu. Ví dụ:

```
int x = 15;
```



Hình 2-1 Biến truy cập vào một vùng nhớ trên bộ nhớ chính

Trong Java, cách duy nhất để đưa dữ liệu vào một biến là sử dụng **câu lệnh gán**. Câu lệnh gán sử dụng **toán tử gán** "=" để gán dữ liệu cho một biến đã khai báo trước đó. Việc khai báo biến thường đi kèm với việc **khai báo kiểu dữ liệu**. Tuy nhiên từ phiên bản Java 10. Việc khai báo biến có thể được thực hiện với từ khóa **var**. Khi đó kiểu của biến sẽ tương ứng với kiểu của dữ liệu được gán cho biến đó, ví dụ:

```
1  var x = 15;
2  var y = 21.2f;
```

Trong Java có thể phân loại thành **ba kiểu biến**:

- Biến cục bộ (**local**)
- Biến thuộc tính (**properties**)
- Biến tĩnh (**static**)

Ví dụ về khai báo và khởi tạo biến hợp lệ trong Java:

```
1  int a, b, c;         // Khai bao ba bien kieu int la a, b, và c.
2  int a = 5, b = 7;  // Vi du ve khoi tao bien
3  byte A = 11;         // Khoi tao mot bien kieu byte ten la A.
4  double pi = 3.14159; // Khai bao va gan mot gia tri cua PI.
5  char nam = 'b';        // Bien nam duoc khoi tao voi gia tri 'a'.
```

Biến cục bộ

Biến cục bộ (biến local) là các biến được khai báo trong các phương thức, hoặc khối lệnh. Các biến cục bộ sẽ được tạo khi các phương thức, khối lệnh này được gọi và sẽ tự động bị hủy sau khi các phương thức, khối lệnh thực hiện xong. Các bổ từ truy cập (**access modifier**) không được sử dụng cho các biến

cục bộ. Các biến cục bộ được thực thi nội bộ trong khối lệnh. Ví dụ biến cục bộ **tuoi** trong đoạn mã dưới đây chỉ có thể hoạt động trong phương thức tuoiCon():

```
public class Test{
   public void tuoiCon(){
      int tuoi = 0;
      tuoi = tuoi + 10;
      System.out.println("Tuoi con la: " + tuoi);
   }

   public static void main(String args[]){
      Test test = new Test();
      test.tuoiCon();
   }
}
```

Kết quả khi chạy chương trình trên:

```
Tuoi con la: 10
```

Các biến cần phải được khởi tạo trước khi sử dụng. Không có giá trị mặc định nào được gán cho các biến cục bộ, vì thế các biến này nên được khai báo và gán một giá trị khởi tạo trước khi sử dụng. Ví dụ sau sử dụng biến **tuoi** mà không khởi tạo nó, vì thế nó sẽ tạo một lỗi ngay khi biên dịch.

```
public class Test{
   public void tuoiCon(){
      int tuoi;
      tuoi = tuoi + 10;
      System.out.println("Tuoi con la : " + tuoi);
   }

   public static void main(String args[]){
      Test test = new Test();
      test.tuoiCon();
   }
}
```

Thông báo lỗi khi biên dịch:

```
Test.java:4: variable number might not have been initialized
tuoi = tuoi + 10;
          ^
1 error
```

Biến thuộc tính

Các **biến thuộc tính (biến properties)** được khai báo trong một lớp, nhưng ở bên ngoài một phương thức. Ví dụ biến **ten**, **hocphi**:

```
public class Student{
   public String ten;
```

```
3     private double hocphi;
4
5     public Student (String tenSV){
6        ten = tenSV;
7     }
8     public void setHocPhi(double hp){
9        hocphi = hp;
10    }
11    public void inThongTin(){
12       System.out.println("Ho va ten: " + ten );
13       System.out.println("Hoc phi: " + hocphi);
14    }
15    public static void main(String args[]){
16       Student sv1 = new Student("Nguyen Van Doan");
17       sv1.setHocPhi(4000);
18       sv1.inThongTin();
19    }
20 }
```

Kết quả khi chạy chương trình trên:

```
1 Ho va ten: Nguyen Van Doan
2 Hoc phi: 4000.0
```

Các **biến thuộc tính** này được tạo khi một đối tượng được tạo bởi sử dụng từ khóa **new** và bị hủy khi đối tượng bị hủy. Các thuộc tính này có thể được sử dụng bởi tất cả các phương thức, phương thức khởi tạo và các khối lệnh trong lớp. Tuy nhiên, tính khả dụng của các thuộc tính này cho các lớp khác phụ thuộc vào việc khai báo các **chỉ định truy cập** (access modifier).

Các đối tượng được cấp phát không gian trong **vùng nhớ heap**, khi đó các thuộc tính của đối tượng cũng được cấp phát không gian cho dù thuộc tính đó có giá trị hay không. Khi các thuộc tính chưa được gán giá trị, nó sẽ có các **giá trị mặc định**. Với các thuộc tính có kiểu là số, giá trị mặc định là 0, với các thuộc tính có kiểu là boolean giá trị mặc định sẽ là false và với đối tượng là null. Giá trị của các biến thuộc tính này có thể được gán trong khi khai báo hoặc trong các phương thức khởi tạo.

Các thuộc tính có thể được truy cập một cách **trực tiếp** bởi việc gọi tên biến bên trong lớp đó. Tuy nhiên, từ các các lớp khác, nó bắt buộc phải được gọi bằng cách sử dụng tên đầy đủ hợp lệ như sau: **"Tên đối tượng.Tên thuộc tính"**.

Biến tĩnh

Các **biến tĩnh** (**biếnstatic**) được khai báo với từ khóa **static** trong một lớp, nhưng ở bên ngoài các phương thức. Các biến static hiếm khi được sử dụng, ngoài việc được khai báo như là các **hằng số**. Các hằng số là các biến, mà được khai báo như là các biến static và bổ sung thêm từ khóa **final**. Các biến hằng số không bao giờ thay đổi từ giá trị khởi tạo của chúng. Các biến static được lưu giữ trong **vùng nhớ static** (là một phần của vùng nhớ heap). Các biến static được tạo khi chương trình bắt đầu và bị hủy khi chương trình kết thúc. Điều này vô cùng quan trọng vì khác với các biến thuộc tính, biến static **không phụ thuộc vào đối tượng**. Giá trị mặc định của biến tĩnh là giống với các biến thuộc tính. Với kiểu số, giá trị mặc định là 0; với kiểu boolean là false và với đối tượng là null. Các giá trị có thể được gán

trong khi khai báo hoặc trong các phương thức. Ngoài ra, các giá trị có thể được gán trong các **phương thức tĩnh**.
Các **biến tĩnh** có thể được truy cập bởi việc gọi tên lớp: "**Tên lớp.Tên biến tĩnh**" hoặc **"Tên đối tượng.Tên biến tĩnh",** tuy nhiên cách sử dụng thứ nhất thì chính xác hơn.

Khi khai báo các biến tĩnh bằng cách sử dụng bổ sung thêm từ khóa **final**, các biến đó sẽ là hằng số và thường được viết ở dạng chữ hoa. Chú ý trong Java có từ khóa **const** nhưng nó không được sử dụng để khai báo hằng số. Ví dụ dưới đây khai báo hằng số FACULTY:

```
1  public class Student{
2     private static double hocphi;
3     public static final String FACULTY = "KhoaCNTT";
4     public static void main(String args[]){
5        hocphi = 4000;
6        System.out.println(FACULTY+"hoc phi trung binh:"+hocphi);
7     }
8  }
```

Chạy chương trình trên, được kết quả sau:

```
KhoaCNTT hoc phi trung binh: 4000
```

Trong ví dụ trên có hai biến tĩnh, trong đó FACULTY là hằng số. Nếu các biến được truy cập từ lớp bên ngoài, các biến tĩnh nên được truy cập như ở dạng: "Tên lớp.Tên biến tĩnh", ví dụ: Student.FACULTY

KIỂU DỮ LIỆU

Kiểu dữ liệu cơ sở

Kiểu **Dữ liệu cơ sở** là kiểu dữ liệu đơn giản nhất trong Java. Tại một thời điểm, một kiểu dữ liệu cơ sở chỉ lưu trữ một giá trị đơn, không có các thông tin khác. Trong Java, có tám kiểu dữ liệu cơ sở

Bảng 2-1 Các kiểu Dữ liệu cơ sở trong Java

Kiểu dữ liệu	Kích thước (bits)	Phạm vi giá trị	Ví dụ
byte	8	-128 đến 127	byte byteNumber = 100;
short	16	-32,768 đến 32,767	short shortNumber = 10000;
int	32	-2,147,483,648 đến 2,147,483,647	int intNumber = 100000;
long	64	-9,223,372,036,854,775,808 đến 9,223,372,036,854,775,807	long longNumber = 100000l;

float	32	1.4E-45 đến 3.4028235E38	float floatNumber = 9.08f;
double	64	4.9E-34 đến 1.7976931348623157E308	double doubleNumber = 9.08;
boolean	1	true/false (mặc định là false)	boolean boolValue = true;
char	16	0 đến 65,535	char charValue = 'a';

Các kiểu dữ liệu như **byte**, **short**, **int long** được sử dụng để lưu trữ các số nguyên. Kích thước và phạm vi giá trị của các biến này được thể hiện trong bảng trên.

Kiểu dữ liệu **boolean** được sử dụng để lưu trữ giá tri đúng hoặc sai (**true** hoặc **false**). Trong khi kiểu dữ liệu **char** được sử dụng để lưu trữ các ký tự. Khác với C, kiểu char sử dụng 16 bits, vì vậy nó có thể biểu diễn được 65,535 ký tự khác nhau.

Bất kỳ ký tự số nào có chứa dấu thập phân đều là ký tự kiểu **double**. Kiểu double sử dụng 8 bytes tương đương với 64 bits. Trong khi đó, kiểu **float** sử dụng cho kiểu dữ liệu dấu phẩy động, được biểu thị bằng 32 bit, với 1 bit dấu, 8 bit số mũ và 23 bit cho phần số.

Để tạo một chữ kiểu **float**, phải thêm "F" hoặc "f" vào cuối số. Ví dụ: "1.2F" là viết tắt của giá trị 1.2 trong kiểu float. Ví dụ:

```
1  float income;
2  income = 3223.56F;
3  System.out.println(income);
```

Khi chạy chương trình. Kết quả nhận được là:

```
3223.56
```

Kiểu dữ liệu "Gói" (Wrapper)

Trong Java, ứng với mỗi kiểu dữ liệu cơ sở thì chúng ta sẽ có một **kiểu dữ liệu gói** (**Wrapper**). Sở dĩ gọi kiểu dữ liệu này là Wrapper là vì nó "gói" các kiểu **dữ liệu cơ sở** vào trong một đối tượng. Vì vậy, Wrapper là kiểu dữ liệu vừa có thể lưu trữ giá trị đơn và vừa có thêm các phương thức khác. Dưới đây là danh sách các Wrapper class ứng với mỗi kiểu dữ liệu cơ sở:

Bảng 2-2 Tham chiếu dữ liệu cơ sở và lớp Wrapper tương ứng

Kiểu dữ liệu cơ sở	Lớp Wrapper tương ứng
boolean	Boolean
char	Character
byte	Byte

short	Short
int	Integer
long	Long
float	Float
double	Double

Ví dụ, số 8 có thể lưu trữ bởi kiểu dữ liệu cơ sở, tuy nhiên cũng có thể sử dụng lớp gói là Integer. Khi đó nó sẽ được cung cấp thêm các phương thức quan trọng. Ngoài ra các lớp Wrapper cũng có các phương thức hữu ích để làm việc với dữ liệu này.

```
1  public static void main(String[] args) {
2      String str= "45";
3      Integer x = Integer.valueOf(str);
4      System.out.println(Integer.toHexString(x));
5  }
```

Nhập xuất các kiểu dữ liệu

Trong Java, để nhập dữ liệu từ bàn phím, cách phổ biến nhất là sử dụng đối tượng **Scanner**. Bước đầu tiên là cần tạo tạo đối tượng Scanner có tên là in để quét dữ liệu từ **System.in** (dòng dữ liệu từ bàn phím)

```
Scanner in = new Scanner(System.in);
```

Bước tiếp theo sẽ sử dụng các phương thức tương ứng của đối tượng Scanner này để nhập dữ liệu với các kiểu dữ liệu khác nhau. Ví dụ:

- Sử dụng nextInt() để đọc số nguyên
- Sử dung nextDouble() để đọc số thực
- Sử dụng next() để đọc một chuỗi

```
1  int num1;
2  double num2;
3  String str;
4  System.out.print("Enter an integer: ");
5  num1 = in.nextInt();
6  System.out.print("Enter a floating point: ");
7  num2 = in.nextDouble();
8  System.out.print("Enter a string: ");
9  str  = in.next();
```

Tương tự như trong C và C++, trong Java, cũng sử dụng phương thức "**printf**" để xuất các kiểu dữ liệu theo định dạng, ví dụ:

```
System.out.printf("%s, Sum of %d & %.2f is %.2f\n", str, num1, num2, num1+num2)
```

Thực hiện hai đoạn mã trên để nhập vào số, chuỗi và in ra màn hình, kết quả thực hiện sẽ như sau:

```
Enter an integer: 3
Enter a floating point: 2.3
Enter a string: Hello
Hello, Sum of 3 & 2.30 is 5.30
```

CHUỖI KÝ TỰ

Chuỗi là một dãy ký tự liên tục. Trong Java, lớp String được sử dụng để tạo ra chuỗi. Như vậy có thể thấy String không phải là kiểu dữ liệu cơ sở, nó bao gồm nhiều ký tự (char) và được hiểu là kiểu dữ liệu dẫn xuất, được tạo ra từ kiểu dữ liệu cơ sở là ký tự.

Ví dụ về chuỗi ký tự:

- “Xin chào”
- “K”
- “Have fun 😄 ”

Khởi tạo chuỗi ký tự

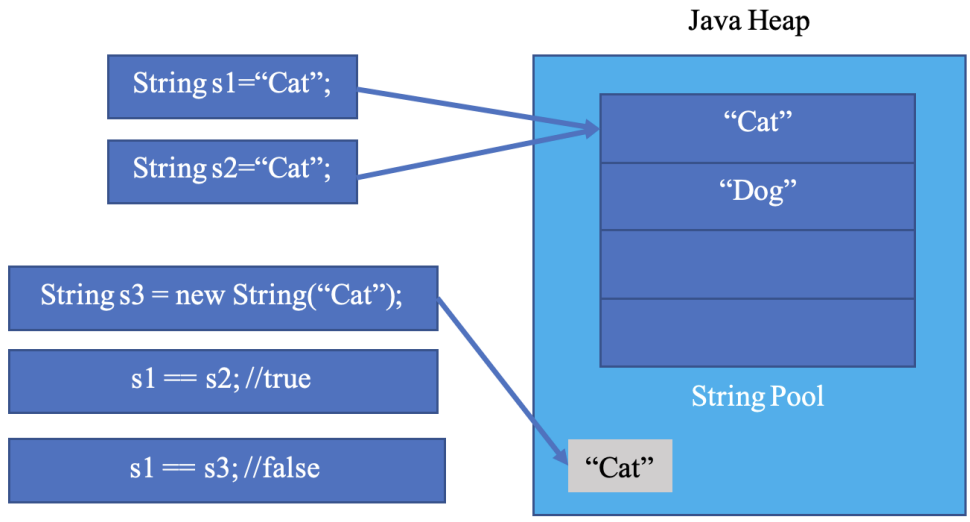
Đối tượng **String** có thể được tạo bằng việc gán với một chuỗi kí tự đặt trong dấu kép. Ví dụ:

```
String s1 = "Cat";
String s2 = "Cat";
```

Java sử dụng String Pool (một vùng nhớ đặc biệt nằm trong vùng nhớ Heap) để lưu trữ các chuỗi nhằm tối ưu hóa việc lưu trữ. Tuy nhiên, theo mặc định thì Java không lưu trữ tất cả đối tượng chuỗi vào String Pool. Đây chính là điểm khác biệt lớn nhất giữa **sử dụng phép gán để tạo String** và sử dụng từ **new để tạo String**. Mỗi khi tạo một hằng chuỗi bằng cách gán giá trị, đầu tiên JVM kiểm tra Pool nơi chứa các hằng chuỗi. Nếu chuỗi đã tồn tại trong Pool, một tham chiếu tới Pool được trả về. Nếu chuỗi không tồn tại trong Pool, một thể hiện của chuỗi mới được tạo và được đặt trong Pool.

Ví dụ, trong trường hợp trên **s1** sẽ tạo chuỗi "Cat" trong String Pool vì chuỗi đó đã có trong String Pool nên biến **s2** sẽ tham chiếu tới chuỗi này. Trong khi đó nếu tạo chuỗi bằng cách sử dụng từ khóa new. Chuỗi được tạo sẽ không được lưu trữ trong String Pool. Ví dụ:

```
String s3 = new String("Cat");
```



Hình 2-2 Cách thức làm việc của String Pool trong bộ nhớ Heap

Các phương thức xử lý chuỗi ký tự

Java cung cấp nhiều phương thức để thao tác trên chuỗi ký tự. Trong phần dưới đây sẽ tìm hiểu chức năng và cách sử dụng của các phương thức này.

Phương thức format() trả về một đối tượng String. Sử dụng phương thức này để tạo một chuỗi đã được định dạng. Ví dụ:

```
String fs;
float floatVar=4.5f;
int intVar = 9;
String stringVar="abc";
fs = String.format("Gia tri cua bien float la " +
                   "%f, trong khi gia tri cua bien integer " +
                   "bien la %d, va chuoi la " +
                   "is %s", floatVar, intVar, stringVar);
System.out.println(fs);
```

Kết quả khi chạy chương trình:

```
Gia tri cua bien float la 4.500000, trong khi gia tri cua bien integer bien la 9, va ch
uoi la is abc
```

Phương thức length() trả về độ dài chuỗi ký tự bằng cách đếm các ký tự trong chuỗi. Cú pháp:

```
int length = tên_chuỗi.length();
```

Ví dụ:

```
public static void main(String[] args) {
    String chuoi;
    int doDai;
    Scanner scanner = new Scanner(System.in);

    System.out.println("Nhập vào chuỗi bất kỳ từ bàn phím: ");
    chuoi = scanner.nextLine();
```

```
8
9       // tính độ dài chuỗi
10      doDai = chuoi.length();
11
12      System.out.println("Chuỗi " + chuoi + " có độ dài = " + doDai);
13  }
```

Ta được kết quả khi chạy chương trình như sau:

```
Chuỗi hello có độ dài = 5
```

Phương thức concat() để nối 2 chuỗi ký tự. Thực tế, trong Java có 2 cách để nối 2 chuỗi ký tự. Cách thứ nhất là chúng ta dùng toán tử + và cách thứ hai là dùng **phương thức concat().**

Cú pháp:

```
String string3 = string1.concat(String string2);
```

Ví dụ:

```
1  public static void main(String[] args) {
2      String chuoi1 = "Happy ", chuoi2 = "new year!";
3      String chuoi3 = chuoi1.concat(chuoi2);
4      System.out.println(chuoi3);
5  }
```

Kết quả khi chạy chương trình:

```
Happy new year!
```

Phương thức charAt() trả về ký tự trong chuỗi theo chỉ số (index(. Một chuỗi là tập hợp các ký tự và ký tự đầu tiên trong chuỗi sẽ có chỉ số là 0. Ví dụ một chuỗi có chiều dài là 10 thì chỉ số của các ký tự trong chuỗi đó sẽ được đánh số từ 0 đến 9. Cú pháp của phương thức charAt():

```
char character = chuoi.charAt(int index);
```

Ví dụ:

```
1  public static void main(String[] args) {
2      String chuoi = "Happy new year!";
3      char character = chuoi.charAt(4); // trả về ký tự có chỉ số là 4 trong chuỗi
4      System.out.println(character);
5  }
```

Kết quả sau khi chạy chương trình:

```
y
```

Phương thức compareTo() có tác dụng so sánh hai chuỗi string1, string2 và trả về kết quả:

- Nếu result = 0 thì hai chuỗi đó bằng nhau.

- Nếu result < 0 thì chuỗi string1 < string2.
- Nếu result > 0 thì chuỗi string1 > string2.

Cú pháp:

```
int result = string1.compareTo(String string2);
```

Ví dụ:

```
1  public static void main(String[] args) {
2      int result;
3      String string1 = "Happy new year!";
4      String string2 = "Happy new year!";
5
6      result = string1.compareTo(string2);
7      if (result == 0) {
8          System.out.println("Chuỗi " + string1 + " = " + string2);
9      } else if (result < 0) {
10         System.out.println("Chuỗi " + string1 + " < " + string2);
11     } else {
12         System.out.println("Chuỗi " + string1 + " > " + string2);
13     }
14 }
```

Kết quả:

```
Chuỗi Happy new year! = Happy new year!
```

Phương thức indexOf() trả về vị trí xuất hiện đầu tiên của chuỗi string2 trong string1. Nếu chuỗi string2 không có trong chuỗi string1 thì kết quả trả về là -1.

Cú pháp:

```
int result = string1.indexOf(String string2);
```

Ví dụ:

```
1  public static void main(String[] args) {
2      int result;
3      String string1 = "Happy new year!";
4      String string2 = "new year!";
5
6      result = string1.indexOf(string2);
7      System.out.println("Vị trí đầu tiên xuất hiện chuỗi " + string2 +
8          " trong chuỗi " + string1 + " = " + result);
9  }
```

Kết quả:

```
Vị trí đầu tiên xuất hiện chuỗi new year! trong chuỗi Happy new year! = 19
```

Phương thức replace() sẽ thay thế ký tự muốn thay thế bằng ký tự khác trong chuỗi. Nếu ký tự cần thay thế không có trong chuỗi string1 thì sẽ trả về chuỗi string1.

Cú pháp:

```
string1.replace(char oldChar, char newChar);
```

Ví dụ:

```
public static void main(String[] args) {
    String string1 = new String("Happy new year!");

    // ký tự thay thế 'l' không có trong chuỗi string1
    System.out.println(string1.replace('l', 'r'));

    // thay thế ký tự 'y' trong string1 thành 'r'
    System.out.println("Chuỗi sau khi thay thế là " +
        string1.replace('y', 'r'));
}
```

Kết quả:

```
Happy new year!
Chuỗi sau khi thay thế là Happy new rear!
```

Phương thức trim() sẽ loại bỏ các khoảng trắng thừa ở đầu và cuối chuỗi string1. Nếu chuỗi đó không có khoảng trắng thừa thì chương trình sẽ trả về chuỗi gốc.

Cú pháp:

```
String string1 = string1.trim();
```

Ví dụ:

```
public static void main(String[] args) {
    String string1 = new String("   Welcome to Vietnam!   ");
    // loại bỏ các khoảng trắng thừa trong chuỗi string1
    string1 = string1.trim();
    System.out.println("Chuỗi sau khi loại bỏ khoảng trắng thừa là " + string1);
}
```

Kết quả:

```
Chuỗi sau khi loại bỏ khoảng trắng thừa là Welcome to Vietnam!
```

Phương thức substring() sẽ tạo một chuỗi con từ vị trí có chỉ số là beginIndex và kết thúc tại vị trí có chỉ số endIndex trong chuỗi cha.

Cú pháp:

```
String chuoiCon = chuoiCha.substring(int beginIndex, int endIndex);
//hoặc từ beginIndex đến hết chuỗi cha
String chuoiCon = chuoiCha.substring(int beginIndex);
```

Ví dụ:

```
public static void main(String[] args) {
    String chuoiCha = new String("Welcome to Java!");
    String chuoiCon1 = chuoiCha.substring(11);
    System.out.println(chuoiCon1);
}
```

Kết quả:

```
Java
```

Bên cạnh những phương thức trên, Java còn có rất nhiều các phương thức khác để làm việc với chuỗi.

TOÁN TỬ

Toán tử là gì

Các **toán tử** được sử dụng để thực hiện các hoạt động trên các **biến** và **giá trị**. Trong ví dụ dưới đây, sử dụng toán tử '+' để cộng hai giá trị với nhau:

```
public class Main {
  public static void main(String[] args) {
    int x = 100 + 50;
    System.out.println(x);
  }
}
```

Mặc dù toán tử '+' thường được sử dụng để cộng hai giá trị với nhau, như trong ví dụ trên, nó cũng có thể được sử dụng để cộng một biến và một giá trị hoặc một biến và một biến khác:

```
public class Main {
  public static void main(String[] args) {
    int sum1 = 100 + 50;
    int sum2 = sum1 + 250;
    int sum3 = sum2 + sum2;
    System.out.println(sum1);
    System.out.println(sum2);
    System.out.println(sum3);
  }
}
```

Kết quả:

```
150
400
800
```

Java chia các toán tử thành các nhóm sau:

- Toán tử số học
- Toán tử gán
- Toán tử so sánh
- Toán tử logic
- Toán tử điều kiện
- Toán tử bitwise

Toán tử số học

Các **toán tử số học** được sử dụng để thực hiện các phép toán thông thường.

Bảng 2-3 Danh sách các toán tử số học

Toán tử	Mô tả	Ví dụ
+	Cộng hai toán hạng	x + y
-	Trừ hai toán hạng	x - y
*	Trừ hai toán hạng	x * y
/	Trừ hai toán hạng	x / y
%	Chia lấy phần dư	x % y
++	Tăng giá trị của toán hạng lên 1 đơn vị	++x
--	Giảm giá trị của toán hạng 1 đơn vị	--x

Toán tử gán

Các **toán tử gán** được sử dụng để gán giá trị cho các biến. Trong ví dụ dưới đây, sử dụng toán tử gán (=) để gán giá trị 10 cho một biến có tên là x:

```
int x = 10;
```

Bảng 2-4 Danh sách các toán tử gán

Operator	Ví dụ	Tương đương
=	x = 5	x = 5
+=	x += 3	x = x + 3

-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3

Toán tử so sánh

Các **toán tử so sánh** được sử dụng để so sánh hai giá trị.

Bảng 2-5 Danh sách các toán tử so sánh

Toán tử	Mô tả	Ví dụ
==	So sánh bằng	x == y
!=	So sánh khác	x != y
>	So sánh lớn hơn	x > y
<	So sánh nhỏ hơn	x < y
>=	So sánh lớn hơn hoặc bằng	x >= y
<=	So sánh nhỏ hơn hoặc bằng	x <= y

Toán tử logic

Các **toán tử logic** được sử dụng để xác định logic giữa các biến hoặc giá trị.

Bảng 2-6 Danh sách các toán tử logic

Toán tử	Mô tả	Ví dụ
&&	Trả về true nếu cả hai toán tử đều đúng	x < 5 && x < 10
||	Trả về true nếu một trong các toán tử là đúng	x < 5 || x < 4
!	Đảo ngược kết quả	!(x < 5 && x < 10)

Biểu thức điều kiện

Bất kỳ ngôn ngữ lập trình tốt nào cũng có một số tính năng nhỏ tiện lợi giúp dễ dàng hơn trong việc viết code. Java có Ternary Operator là **toán tử điều kiện**, đây là một toán tử ba ngôi giúp việc viếc câu lệnh if được gọn hơn. Nó có cấu trúc như sau:

```
Biểu thức? lệnh 1: lệnh 2
```

Máy tính kiểm tra giá trị của **Biểu thức**. Nếu giá trị là true, tiến hành thực thi **lệnh 1**, nếu sai, thực thi **lệnh 2.**

Ví dụ:

```
next = (N % 2 == 0) ? (N/2) : (3*N+1);
```

Câu lệnh trên sẽ gán giá trị N/2 cho next nếu N là chẵn (nghĩa là nếu N % 2 == 0 là true) và nó sẽ gán giá trị (3* N + 1) cho next nếu N là lẻ. (Dấu ngoặc đơn trong ví dụ này không bắt buộc, nhưng chúng làm cho biểu thức dễ đọc hơn).

Xét một ví dụ khác:

```
1  public class Main {
2    public static void main(String[] args) {
3      int time = 20;
4      String result;
5      result = (time < 18) ? "Good day." : "Good evening.";
6      System.out.println(result);
7    }
8  }
```

Kết quả khi chạy:

```
Good evening.
```

Toán tử bitwise

Các **toán tử bitwise** được sử dụng để làm việc với giá trị bít đơn lẻ của một số. Toán tử này có thể sử dụng cho các kiểu dữ liệu số nguyên như int, short, char, long,...

Toán tử	Mô tả	Ví dụ
~	Đảo ngược giá trị của bit	~ x
&	Trả về 1 nếu là 2 bit 1	x & y
^	Trả về 1 nếu hai bít khác nhau	x ^ y
|	Trả về 1 nếu 1 trong 2 bit là 1	x | y
<<	Dịch các bit trang trái	x << 2
>>	Dịch các bit sang phải	x >> 2

Ví dụ:

```
int x = 20; //10100
int y = 4;  //00100
System.out.println("~x = " + ~x);
System.out.println("x & y = " + (x & y));
System.out.println("x ^ y = " + (x ^ y));
System.out.println("x | y = " + (x | y));
System.out.println("x << y = " + (x << y));
System.out.println("x >> y = " + (x >> y));
```

Kết quả:

```
~x = -21
x & y = 4
x ^ y = 16
x | y = 20
x << y = 320
x >> y = 1
```

THỰC HÀNH

Xác định thông tin về các kiểu dữ liệu

Bài thực hành này sẽ viết mã lệnh để in độ dài tối đa, tối thiểu và độ dài bit của các kiểu dữ liệu cơ bản. Bắt đầu bằng việc tạo lớp TypesMinMax với phương thức main()

```
public class TypesMinMax {
   public static void main(String[] args) {
   }
}
```

Trong phương thức main(), in giá trị Max và Min của kiểu int:

```
System.out.println("int(min) = " + Integer.MIN_VALUE);
System.out.println("int(max) = " + Integer.MAX_VALUE);
System.out.println("int(bit-length) = " + Integer.SIZE);
```

In giá trị Max và Min của kiểu byte:

```
System.out.println("byte(min) = " + Byte.MIN_VALUE);
System.out.println("byte(max) = " + Byte.MAX_VALUE);
System.out.println("byte(bit-length)=" + Byte.SIZE);
```

In giá trị Max và Min của kiểu short:

```
System.out.println("short(min) = " + Short.MIN_VALUE);
System.out.println("short(max) = " + Short.MAX_VALUE);
```

```
3  System.out.println("short(bit-length) = " + Short.SIZE);
```

In giá trị Max và Min của kiểu long:

```
1  System.out.println("long(min) = " + Long.MIN_VALUE);
2  System.out.println("long(max) = " + Long.MAX_VALUE);
3  System.out.println("long(bit-length) = " + Long.SIZE);
```

In giá trị Max và Min của kiểu char (16-bit character hoặc 16-bit số nguyên không dấu):

```
1  System.out.println("char(min) = " + (int)Character.MIN_VALUE);
2  System.out.println("char(max) = " + (int)Character.MAX_VALUE);
3  System.out.println("char(bit-length) = " + Character.SIZE);
```

In giá trị Max và Min của kiểu float:

```
1  System.out.println("float(min) = " + Float.MIN_VALUE);
2  System.out.println("float(max) = " + Float.MAX_VALUE);
3  System.out.println("float(bit-length) = " + Float.SIZE);
```

In giá trị Max và Min của kiểu double:

```
1  System.out.println("float(min) = " + Float.MIN_VALUE);
2  System.out.println("float(max) = " + Float.MAX_VALUE);
3  System.out.println("float(bit-length) = " + Float.SIZE);
```

Thực hiện chương trình, kết quả khi chạy sẽ cho biết thông tin về kiểu Dữ liệu cơ sở:

```
1  int(min) = -2147483648
2  int(max) = 2147483647
3  int(bit-length) = 32
4  byte(min) = -128
5  byte(max) = 127
6  byte(bit-length)=8
7  short(min) = -32768
8  short(max) = 32767
9  short(bit-length) = 16
10 long(min) = -9223372036854775808
11 long(max) = 9223372036854775807
12 long(bit-length) = 64
13 char(min) = 0
14 char(max) = 65535
15 char(bit-length) = 16
16 float(min) = 1.4E-45
17 float(max) = 3.4028235E38
18 float(bit-length) = 32
```

So sánh chuỗi

Tạo lớp **StringTest** với phương thức main()

```
public class StringTest {
   public static void main(String[] args) {
   }
}
```

Thực hiện khai báo và so sánh các chuỗi

```
String s1="Cat";
String s2="Cat";
String s3=new String("Cat");
String s4="Dog";
String s5="CAT";
System.out.println(s1==s2);
System.out.println(s1==s3);
System.out.println(s1.equals(s2));
System.out.println(s1.equals(s3));
System.out.println(s1.equals(s5));
System.out.println(s1.equalsIgnoreCase(s5));
```

Kiểm tra kết quả in ra:

```
true
false
true
true
false
true
```

Toán tử

Tạo lớp **OpTest** sau đó khai báo và khởi tạo các biến:

```
int age = 18;
double weight = 71.23;
int height = 191;
boolean married = false;
boolean attached = false;
char gender = 'm';
```

Kiểm tra các toán tử quan hệ và logic

```
System.out.println(!married && !attached && (gender == 'm'));
System.out.println(married && (gender == 'f'));
System.out.println((height >= 180) && (weight >= 65) && (weight <= 80));
System.out.println((height >= 180) || (weight >= 90));
```

Thực hiện chương trình, kiểm tra kết quả:

```
true
false
```

```
3 true
4 true
```

CÂU HỎI ÔN TẬP LÝ THUYẾT

1. Đâu là kiểu dữ liệu phù hợp cho số 20.15

- A. long
- B. short
- C. Integer
- D. float

2. Toán tử gán được sử dụng để

- A. Gán một giá trị cho một biến
- B. Gán một câu lệnh cho một phương thức
- C. Gán tên cho một giá trị
- D. Gán một giá trị cho một biểu thức

3. Kiểu dữ liệu byte có phạm vi

- A. 0 đến 256
- B. Từ 0 đến 128
- C. Từ 0 đến 8
- D. Từ -128 đên 127

4. Một biến trong java có thể chứa

- A. Ký tự $
- B. Ký tự @

C. Ký tự -

D. Ký tự %

5. Từ khóa/ lệnh break được dùng để

A. Phá vỡ một khối lệnh

B. Bỏ qua một lượt lặp

C. Thoát ra khỏi vòng lặp hoặc khối lệnh switch

D. Kết thúc một lệnh rẽ nhánh

6. Đâu không phải là một lớp Wrapper (Wrapper class)

A. Integer

B. Byte

C. String

D. Float

7. Khai báo một hằng số sử dụng từ khóa

A. static

B. final

C. const

D. constant

8. Một biến double sử dụng bao nhiêu byte?

A. 2

B. 4

C. 8

D. 16

9. Đối tượng Scanner được sử dụng để

A. Tìm kiếm thông tin từ các biến

B. Quét các lỗi có thể xảy ra

C. Quét dữ liệu từ một stream đầu vào

D. Lấy thông tin từ mảng

10. Tên một gói có thể chứa

A. Dấu chấm .

B. Dấu chấm than !

C. Dấu ngoặc nhọn {}

D. Dấu phần trăm %

11. Hãy cho biết kết quả khi thực hiện đoạn mã sau:

```
1  int a = -10;
2  int b = a < 0 ? -a : a;
3  System.out.println("a = " + b);
```

A. b = -10

B. b = 10

C. b = 0

D. b = 1

12. Hãy cho biết kết quả thực hiện đoạn mã sau:

```
for (int i = 0; i < 26; i++) {
    char letter = (char) ('a' + i);
    System.out.println(letter);
}
```

- A. Hiển thị 25 chữ cái a
- B. Hiển thị 26 chữ cái a
- C. Hiển thị bảng chữ cái
- D. Xuất hiện một lỗi biên dịch

13. Hãy cho biết kết quả thực hiện đoạn mã sau:

```
short high = Short.MAX_VALUE;
short low = Short.MIN_VALUE;
System.out.println(high - low);
```

- A. 32767
- B. 0
- C. 65534
- D. 65535

14. Hãy cho biết kết quả thực hiện đoạn mã sau:

```
String strObj = new String("NEU");
String str = "NEU" ;
System.out.println(strObj.equals(str));
```

- A. in ra: true
- B. in ra: false
- C. Lỗi biên dịch
- D. in ra chữ NEU

15. Hãy cho biết kết quả thực hiện đoạn mã sau:

```
1  String strObj = new String("NEU");
2  String str = "NEU" ;
3  System.out.println(str == strObj);
```

- A. in ra: true
- B. in ra: false
- C. Lỗi biên dịch
- D. in ra chữ NEU

BÀI TẬP TỰ THỰC HÀNH

Tính thuế

Thuế bán hàng của mỗi một sản phẩm là 8,25%. Viết chương trình nhập vào **giá một sản phẩm** ban đầu và in **số tiền phải trả cho phần thuế** và **giá sau khi tính thuế** của sản phẩm.

Bài toán trả tiền làm thêm giờ

Viết chương trình yêu cầu người dùng nhập vào **số giờ làm việc** và **số tiền trả cho mỗi giờ**. Sau đó tính toán để in ra **số tiền phải trả** cho nhân viên đó.

Chuyển đổi Centimeter - inch

Biết rằng 2,54 cm = 1 inch. Hãy viết một chương trình chuyển một số từ **cm** thành **inch**. Người dùng sẽ nhập độ dài theo **cm**, hãy hiển thị kết quả sau khi quy đổi sang đơn vị **inch**.

TÀI LIỆU THAM KHẢO

[1] Core Java: Fundamentals (2021) , Cay Horstmann (Oracle Press Java)

[2] Head First Java: A Brain-Friendly Guide (2022), Kathy Sierra, O'Reilly Media

[3] Java OOP Done Right: Create object oriented code you can be proud of with modern Java Paperback (2019), Mr Alan Mellor, Mellor Books

[4] Murach's Java Programming (5th Edition) (2017), Joe Murach, Mike Murach & Associates

[5]. Java for Absolute Beginners Learn to Program the Fundamentals the Java 9+ Way

[6]. Modern Java Recipes: Simple Solutions to Difficult Problems in Java 8 and 9 (2017), by Ken Kousen, O'Reilly Media

[7] Effective Java (2018), Joshua Bloch, Addison-Wesley Professional