JAVA PROGRAMMING

PISCES KIBO

MỤC LỤC

Bài 1: Chương trình đầu tiên của Java

1. **Công thức cơ bản của câu lệnh Java:**

```
public class File_name {
    public static void main(String[]args) {
        System.out.println("Hello Java");
    }
}
```

- System.out.println(): câu lệnh in ra màn hình có xuống dòng
- System.out.print(): câu lệnh in ra màn hình không xuống dòng
- System.err.println(): in ra màn hình câu lệnh lỗi màu đỏ
- System.out.printf(local, format, arg1, arg2, …): format dữ liệu bên trong
    - local: nếu khác null sẽ được tự động định dạng theo khu vực (có thể null)
    - format: quy chuẩn định dạng đầu ra cho các đối số
    - arg: các đối số cần định dạng
    - Bộ định dạng format ký tự cho printf():
        - %c : ký tự theo dữ liệu arg
        - %C : ký tự HOA theo dữ liệu arg
        - %s : chuỗi theo dữ liệu arg
        - %S : chuỗi in HOA theo dữ liệu từ arg
        - %t : định dạng ngày giờ
        - %d : số thập phân/số nguyên
        - %d: số thực
        - \n : tự động xuống dòng
        - \t : thụt đầu dòng
        - \” string \” : ngoặc kép trong chuỗi

Vd: a = 1, b = 2
System.out.printf(“%d + %d = %d”, a, b, a+b)
➔ 1 + 2 = 3

2. **Một số chú ý khi code Java:**
    - Tên file trùng với tên class
    - Tên file có định dạng là “filename.java”
    - Sau mỗi câu lệnh cần có dấu “;”
    - Nhét chuỗi cần in trong dấu ngoặc kép: “Chuỗi”

    - *Quy chuẩn cấu trúc Code Java (Java Style):*

- Ngoài file:
  - Tên package viết thường
  - Tên class (tên file) viết liền và viết hoa các chữ cái đầu tiên

- Trong file:
  - Tên hàm: viết liền, chữ cái đầu của từ đầu tiên viết thường, các từ sau có chữ cái đầu viết Hoa
  - Tên biến: viết liền, chữ cái đầu của từ đầu tiên viết thường, các từ sau có chữ cái đầu viết Hoa
  - Tên HẰNG: tất cả đều viết in HOA

3. Các loại chú thích Comment:

- Chú thích trên một dòng: *// đây là chú thích 1 dòng*
- Chú thích nhiều dòng:

/* * *đây là chú thích nhiều dòng* */	/* đây là chú thích nhiều dòng */

- Ghi chú dạng Javadocs:

```
/**
* @param TenBien
* Chú thích ý nghĩa các tham số
*/
```

Bài 2: Biến và kiểu dữ liệu cơ bản

Kiểu	Mô tả	Kích cỡ	Tối thiểu	Tối đa	Lớp bao
byte	số nguyên một byte	8 bit	-128	127	Byte
short	số nguyên ngắn	16 bit	-2^15	2^15-1	Short
int	số nguyên	32 bit	-2^31	2^31-1	Integer
long	số nguyên dài	64 bit	-2^63	-2^63-1	Long
float	kiểu thực với độ chính xác đơn	32 bit	IEEE754	IEEE754	Float
double	Double-precision floating point	64 bit	IEEE754	IEEE754	Double
char	kiểu kí tự	16 bit	Unicode 0	Unicode 2^16-1	Character
boolean	kiểu logic	true hoặc false	-	-	Boolean
void	-	-	-	-	Void

Kiểu dữ liệu nguyên thủy

1. **Kiểu số nguyên:**
   - byte (8 bit): [-128, 127]
   - short (16 bit): [-327668, 32767] → [-2^15, 2^15 -1]
   - int (32 bit): [-217483648, 217483647] → [-2^31, 2^31]
   - long (64 bit): [-9223372036854775808, 9223372036854775807] → [-2^63,2^63-1]
     vd:
     byte x = 10;
     short s = 30;

int i = 3000;
long l = 500000L;

2. **Kiểu số thực:**
   - float (32 bit) → 0.0f
   - double (64 bit) → 0.0d

   vd:
   float f = 30.044f;
   double d = 300.44d;

3. **Kiểu Boolean: (true/false)**
   - Có 2 giá trị true/false
   - Giá trị mặc định là false

   vd:
   boolean t = true;
   boolean f = false;

4. **Kiểu ký tự và chuỗi:**
   - char (16 bit) → ký tự
   - String → chuỗi
     - Khai báo chuỗi trong dấu ngoặc kép
     - Có thể cộng nối hai chuỗi lại với nhau
     - String có thể kết hợp cùng với nhiều kiểu dữ liệu khác nhau mà không cần ép kiểu dữ liệu → định dạng kiểu chuỗi
     - Nếu muốn in ra kết quả của phép tính thì ta để phép tính trong dấu ngoặc tròn "(phép tính)", nếu không sẽ mặc định kiểu dạng là chuỗi
       Vd: System.out.println("a + b =" + (a+b))

     - Có thể kết hợp các kiểu dữ liệu khác nhau ngay trong System.out.prinln()

   Vd:
   char a = 'A'; → kiểu ký tự
   String b = "chuỗi"; → kiểu chuỗi

5. **Gán biến kiểu hằng số:**
   - final <type> biến = hằng số;
     vd: final int HANG_SO = 314;

   - Hằng số được khai báo ngoài hàm main chính:
     public static final <type> Tên_hằng_số = <value>;

   - Giá trị hằng số không thay đổi được trong các hàm

6. **Cách khai báo biến:**

- Công thức gán biến: *<type> <name_para> = <value>;*
- Công thức gọi biến: *<type> <name_para>;*
    - type là kiểu giá trị
    - value là giá trị của biến
    - name_para là tên biến
- Có thể khai báo nhiều biến cùng dòng: *<type> name1, name2;*

+ Một số chú ý khi đặt tên biến:

- Cách đặt biến khác nhau dựa vào chữ cái HOA và thường
- Tên biến được phép bắt đầu với A-Z, a-z, $, _
- Không được đặt tên biến trùng với các ký tự, từ khóa đặc biệt, có khoảng trắng
- Các hằng số nên viết hoa: HANG_SO

Bài 3: Các phép toán tử cơ bản

1. **Toán tử số học:**
   <type> A = a;
   <type> B = b;
   ⇨ <type> phep_tinh = phép tính giữa A với B;

Toán tử	Miêu tả	Phép tính
+	Phép cộng	<type> tong = A + B
-	Phép trừ	<type> hieu = A – B
*	Phép nhân	<type> tich = A * B
/	Phép chia	<type> thuong = A / B
%	Phép chia lấy phần dư	<type> du = A % B
++	Tăng giá trị lên 1 đơn vị	A++ hoặc ++A
--	Giảm giá trị xuống 1 đơn vị	A - - hoặc - -A

   - Sự khác nhau giữa A++ (A - -) và ++A (- - A):
     - A++ : in ra rồi mới tăng (Postfix)
     - ++A : tăng xong mới in ra (Prefix)
     - A - - : in ra rồi mới giảm (Postfix)
     - - - A : giảm rồi mới in ra (Prefix)

   ⇨ Việc đưa toán tử phía sau có nghĩa là sau khi thực hiện công việc trên rồi mới tăng/giảm giá trị, việc đưa lên trước thì nó sẽ tăng/giảm giá trị rồi mới thực hiện tiếp công việc

2. **Tự động nâng kiểu dữ liệu và ép kiểu dữ liệu trong Java:**
   <type1> tên_biến1 = value1;
   <type2> tên_biến2 = value2;
   <type3> tên_biến3 = value3;
   ⇨ <type> ketqua = phép tính;

   - Nếu <type1>, <type2>, <type3> khác nhau thì sẽ thực hiện các phép tính theo kiểu dữ liệu <type> lớn nhất

   Vd:

```
// Khai báo biến
byte b = 3;
short a = 67;
int i = 100;
long k = 400L;

// Nâng kiểu dữ liệu lớn nhất của các biến
long tongL = k + i + a + b;           // Kiểu dữ liệu đó thành kiểu long
```

```
int tongI = i + a + b;                    // Kiểu dữ liệu đó thành kiểu int
```

- Ta có thể ép kiểu dữ liệu khi thực hiện:
  - <typeM> tên_biếnM = (<typeM>) tên_biến3 + ...

Vd:

```
// Cách ép kiểu dữ liệu
int tongK = (int) k + i + a;
float tongF = (float) thucDouble + thucFloat;
```

- Ta cũng có thể ép kiểu dữ liệu giữa số nguyên và số thực
- Nếu muốn chuyển số nguyên sang số thực có thể khai báo số thập phân
  VD: a → integer; a.0 → double

- Chú ý: con ép kiểu về cha được nhưng mà cha không ép kiểu được về con

3. Toán tử quan hệ so sánh:

- Kiểm tra giá trị của 2 biến A và B
- Đầu ra cho kiểu dữ liệu true/false

Toán tử	Miêu tả	Ví dụ
==	Bằng nhau	A == B
!=	Khác nhau	A != B
>	Lớn hơn	A > B
<	Nhỏ hơn	A < B
>=	Lớn hơn hoặc bằng	A >= B
<=	Nhỏ hơn hoặc bằng	A <= B

4. Toán tử LOGIC:

- Các toán tử đưa ra giá trị đúng/sai trong các mệnh đề với nhau

Toán tử	Miêu tả	Ví dụ
!	Toán tử NOT	!A
&&	Toán tử AND	A && B
||	Toán tử OR	A || B

5. Toán tử gán:

- Gán giá trị vế trái thành phép tính của vế trái với vế phải

Toán tử	Miêu tả	Cụ thể
=	Gán giá trị	A = B
+=	a += b	a = a + b
-=	a -= b	a = a – b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Bài 4: Cách tạo hàm và sử dụng Java

1. **Cấu trúc của tao hàm chung:**

```
public class File_name {
      // Hàm main mặc định chạy chương trình
      (Gọi các hàm tự tạo vào trong hàm main)
      public static void main(String[]args) {
            // Gọi hàm ngoài trong hàm main
            <type> tên_biến = tên_hàm_ngoài(đối số);
            System.out.println(tên_biến);

            // Lấy dữ liệu trực tiếp từ hàm ngoài
            System.out.println(tên_hàm_ngoài(đối số));        // Nếu là return
            tên_hàm_ngoài(đối số);        // Nếu là System.out.println()
      }

      Khởi tạo hàm 1
      Khởi tạo hàm 2
      ...
}
```

- Luôn hiển thị kết quả hàm ngoài ra màn hình bằng cách gọi các hàm ngoài vào trong hàm main chính

2. **Cách khởi tao các hàm ngoài:**

a) Loại 1: Khởi tạo hàm dựa trên đối số:

```
public static <type> tên_hàm(đối số 1, đối số 2, …) {
        Khai báo các biến;
```

```
        return kết_quả;
}
```

- Có thể không cần truyền đối số đầu vào

b) Loại 2: Khởi tạo hàm không dựa trên đối số:

```
public static void tên_hàm() {
        System.out.println(Nhập chuỗi);
}
```

- Kiểu dữ liệu trả về không phụ thuộc vào kiểu dữ liệu

3. Khai báo in ra các hàm trong hàm main chính:

+ Giá trị biến của từng hàm là khác nhau mặc dùng có cùng tên biến
+ Cách gọi hàm ngoài trong hàm main:
    - <type> tên_biến = tên_hàm_ngoài(đối số);
    - System.out.println(tên_biến);

+ Các cách hiển thị trực tiếp các hàm ngoài:
    - Nếu trong hàm ngoài chứa kiểu return:
      System.out.println(tên_hàm_ngoài(đối số));
    - Nếu trong hàm ngoài chứa kiểu System.out.println():
      tên_hàm_ngoài(đối số);

Bài 5: Khởi tạo đối tượng từ ngoài File

1. **Các hàm ở ngoài file chính:**
    - Không chứa hàm main mặc định trong file
    - Áp dụng cách khởi tạo hàm ngoài hàm main chính
    - Đặt tên cho đối tượng

```
package JavaCode;

public class Tên_đối_tượng {

	// Khởi tạo hàm 1
	public static <type> Tên_hàm1(<type> biến) {
		return Kết_quả;
	}

	// Khởi tạo hàm 2
	public static <type> dienTich(đối số) {
		return Kết_quả;
	}

}
```

2. **Phương thức trong hàm chính:**
    - Có thể khai báo hằng số phía ngoài hàm main
    - Mọi phương thức đều thực hiện trong hàm main
    - Khởi tạo biến cho toàn bộ chương trình
      <type> tên_biến = <value>;

    - Gọi đối tượng từ bên ngoài vào trong hàm main chính
      <type> Ketqua = Doituong.hàmĐT(tên biến 1, tên biến 2, …);

```
package JavaCode;

public class Tên_filename {
	// Có thể gọi hằng số

	// Hàm main chính của chương trình
	public static void main(String[] args) {
		// Khởi tạo biến cho chương trình
```

```
            <type1> tên_biến1 = <value1>;
            <type2> tên_biến2 = <value2>;
             ……

            // Gọi đối tượng từ bên ngoài
            <type> Ketqua1 = Đốitượng1.hàmĐT1(đối số 1, đối số 2, …);
            <type> Ketqua2 = Doituong2.hàmĐT2(tên biến 1, tên biến 2, …);
      }

}
```

3. **<u>Gói Package và Import trong Java:</u>**

+ Thư mục lớn gồm nhiều các projects khác nhau: pro1, pro2, …
+ Sử dụng import tại hàm main chính của project chính để xuất thư viện của các thư mục projects khác ở phía ngoài

```
package Tên_ProjectFolder_mainchinh;
import Tên_projectfolder_phụ_ngoài.Filename;
public static Tên_filemain {
        public static void main(String[] args) {
                // Gọi các đối tượng xử lý
                Filename1.nameham();
                Filename2.nameham();
        }
}
```

- Filename1, Filename2 là tên các file trong project phụ
- Khởi gọi các đối tượng ngoài trong hàm main chính với cấu trúc:
  Filename.nameham();

Bài 6: Nhập xuất giá trị đầu vào

1. **Dữ liệu đầu vào:**

```java
package JavaCode;

// Thư viện đầu vào Scanner
import java.util.Scanner;

public class File_name {

	public static void main(String[] args) {
		// Tạo giá trị đầu vào bất kỳ từ thư viện
		Scanner scan = new Scanner(System.in);

		// Nhập một giá trị đầu vào
		System.out.println("Vui lòng nhập giá trị:");
		int x = scan.nextInt();			//Nhập giá trị của biến vào
		System.out.println("=> Giá trị của x là: " + x);


		// Công thức nhập giá trị đầu vào từ bàn phím
		/*
		 * import java.util.Scanner --> phía ngoài class
		 *
		 * Scanner scan = new Scanner(System.in);
		 * <type> tên_biến = scan.next<Type>();
		 */

	}

}
```

- Import thư viện Scanner từ ngoài chương trình class
  import java.util.Scanner;
- Tạo giá trị đầu vào bất kỳ từ thư viện:
  Scanner scan = new Scanner(System.in);
- Nhập giá trị của biến:
  <type> tên_biến = scan.next<Type>();

- Cách viết rút gọn:

<type> NameBien = new Scanner(System.in).next<Type>();

Vd1: char NameBien = new Scanner(System.in).nextLine().charAt(0);
Vd2: int NameBien = new Scanner(System.in).nextInt();

2. **Công thức nhập giá trị đầu vào:**
   - ***import java.util.Scanner*** → gọi thư viện từ phía ngoài chương trình
   - Phía trong chương trình hàm main mặc định:
     - ***Scanner scan = new Scanner(System.in);***
     - ***<type> tên_biến = scan.next<Type>();***
   - Chú thích: "scan" là tên biến chính, có thể đặt tên khác

Vd: int x = scan.nextInt();

3. **Các kiểu dữ liệu đầu vào:**

+ nextByte(); → số nguyên 1
+ nextShort(); → số nguyên 2
+ nextInt(); → số nguyên 3
+ nextLong(); → số nguyên 4

+ nextFloat(); → số thực 1
+ nextDouble(); → số thực 2

+ nextBoolean(); → biến boolean

+ nextLine(); hoặc next(); → kiểu chuỗi String

4. **Chú ý:**
   - Nếu đầu vào bàn phím khác kiểu dữ liệu với nhau thì cần khai báo lại các biến "scan" khác cho các kiểu dữ liệu đó

```
Scanner scan1 = new Scanner(System.in);
Scanner scan2 = new Scanner(System.in);
…
```

Bài 7: Các câu lệnh điều kiện rẽ nhánh

1. **Câu lệnh điều kiện IF – ELSE IF - ELSE:**

```
if ( [Biểu thức điều kiện] ) {
        #Nếu đúng thực hiện bước này;
} else if {
        #Nếu đúng thực hiện bước này;
} else if {
        …
} else {
        # Nếu đúng thực hiện bước này
}

#Nếu sai thực hiện bước này
```

- Câu lệnh điều kiện rẽ nhánh chạy chương trình trong các trường hợp khác nhau
- Kiểm tra biểu thức điều kiện, nếu đúng thì biểu thức bên trong câu lệnh điều kiện được thực thi

Vd:

```
if (y == 5) {
      System.out.println("Giá trị của y bằng 5");
else if (y > 5) {
      System.out.println("Giá trị y lớn hơn 5");
else {
      System.out.println("Giá trị của y nhỏ hơn 5");
}
```

2. **Câu điều kiện Switch – Case – Default:**

CẤU LỆNH CƠ BẢN	BIẾN THỂ SWITCH - CASE

switch (tên biến variable) {
case <value1>:
// Thực thi câu lệnh bên trong
break;
case <value2>:
// Thực thi câu lệnh bên trong
break;
…
default:
	switch (tên biến variable) {
	case <value1>:
	case <value2>:
	// Thực thi câu lệnh bên trong (khi <value1> và <value2> cho cùng kết quả
	break;
	…
	default:
	// Nếu không có trong các trường hợp trên thì chạy câu lệnh này

```
// Nếu không có trong các trường hợp trên thì chạy câu lệnh này
```

- Nếu không có break thì chương trình sẽ chạy lần lượt tất cả các trường hợp ứng với từng giá trị của biến
- Khi thấy có case đúng, khối lệnh trong case đó sẽ được chạy
- Câu lệnh Break: phá vỡ vòng lặp, thoát ngay ra khỏi vòng lặp
- Khối lệnh default không bắt buộc xuất hiện trong cấu trúc switch - case

Vd:

```
switch(x) {
	case 1:
		System.out.println("x = 1");
		break;
	case 2:
		System.out.println("x = 2");

	case 3: System.out.println("x = 3");

	……

	default:
		System.out.println("x là default");
}
```

3. Toán tử điều kiện ? và : :

Kq = (kiểm tra điều kiện) ? (kq1 nếu đúng) : (kq2 nếu sai)

- Dùng để check điều kiện và gán biến
  - ? tương đương với phần của if
  - : tương đương với phần của else

Vd:

```
package JavaCode;

// Toán tử điều kiện ? :

public class B14_ToanTuDieuKien {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int tong = 0;
		int x = 5;
```

```
        // Cách viết tắt khác
        /*Dùng để check điều kiện và gán biến
         * ? tương đương với phần của if
         * : tương đương với phần của else
         *
         * kq = (kiểm tra điều kiện) ? (kq1 nếu đúng) : (kq2 nếu sai)
         */

        tong = (x > 5) ? (tong + 1) : (tong - 1);     //Kiểu int
        String s = (x < 5) ? "Xin chào" : "Goodbye";    //Kiểu String

    }

}
```

Bài 8: Khai báo mảng dữ liệu Java

1. **Cách khai báo mảng 1 chiều:**
    - Mảng trong Java có định dạng kiểu: mang = { }
    - Mảng là tập hợp các giá trị của biến ứng với index → biến cùng kiểu dữ liệu

a) Mảng cố định:

+ Có sẵn các phần tử trong mảng như {a, b, c, …}
+ Công thức mảng cố định: (có 2 cách)

```
<type>[ ] tên_mảng = {arg1, arg2, …};
<type> tên_mảng[ ] = {val1, val2, …};
```

- <type> : kiểu dữ liệu chung của các phần tử trong mảng
- Tên_mảng : đặt tên mảng tùy chọn
- arg và val là các giá trị của biến được truyền vào sẵn

b) Mảng thông thường 1 chiều:

```
+ Cách 1: <type>[ ] tên_mảng = new <type>[số phần tử];
+ Cách 2: <type> tên_mảng[ ] = new <type>[kích thước mảng];
```

- Khuyến khích dùng cách 1
- Mảng được sử dụng sau và được truyền bằng kích cỡ cho sẵn
- Gán giá trị của mảng bằng index: ***tên_mảng[i] = <value>;***

2. **Truy xuất dữ liệu trong mảng:**

+ Các phần tử được truy xuất bằng toán tử [i] với i là index
+ Truy xuất phần tử cuối cùng bằng cách mang[-1] với index i = -1
+ Phần tử đầu tiên luôn có index i = 0 tức mang[0]

- Gọi giá trị của mảng bằng: ***tên_mảng[i]***
- Phương thức kiểm tra độ dài mảng: ***tên_mảng.length***

Bài 9: Các cấu trúc vòng lặp

1. **Vòng lặp While:**

+ Câu trúc vòng lặp while:

```
<type> tên_biến = <value>;
while (<điều kiện>) {
      # Thực thi các câu lệnh bên trong;
      tên_biến = tên_biến + step;
}
```

  - Lặp lại với số lần không biết trước, nếu không có thêm biến chạy ngoài thì sẽ chạy vô hạn lần
  - Nên có thêm biến đếm chạy trọng vòng lặp while

- Ví dụ vòng lặp in số:

```
int i = 0;
while (i < 5) {
    System.out.println(i);
     i = i + 1;
}
       System.out.println("...");
```

→0, 1, 2, 3, 4, …

- Ví dụ về vòng lặp mảng:

```
int[] x = {1,2,3,4,5};
int index = 0;
while (index < x.length) {
    System.out.println(x[index]);
     index++;     // tương đương index += 1
}
```

→ 1, 2, 3, 4, 5

2. **Vòng lặp Do – While:**

+ Chạy chương trình trước rồi mới kiểm tra điều kiện

```
<type> nameValue = 0;
do {
            // Khối lệnh thực thi
            nameValue = nameValue +
1;
} while (<điều kiện>);
```

Vd:

Vòng lặp số	Vòng lặp mảng

int i = 0;
// Chạy trước và kiểm tra điều kiện sau
do {
System.out.println(i);
i = i + 1;
} while (i < 5);
	int[] x = {1,2,3,4,5,6};
	int index = 0;
	do {
	System.out.println(x[index]);
	index++;
	} while (index < x.length);

3. **Vòng lặp For:**

+ Công thức:

```
for (int i = start; i < end; I += step) {
    // Thực thi khối lệnh lặp
}
```

- Vòng lặp for thiếu:

```
int i = start;
for (; i < end; ) {
        // Thực thi khối lệnh lặp
        i += step;
}
```

VD:

Loại 1: i++ => tăng lên 1	`for (int i = 0; i < 4; i++) {`
	`System.out.println(i);`
	`}`

Loại 2: i-- => giảm đi 1	`for (int i =5; i > 0; i--) {`
	`System.out.println(i);`
	`}`

Loại 3: Vòng lặp mảng	`int[] x1 = {1,2,3,4,5};`
	`for (int i = 0; i < x1.length; i++) {`
	`System.out.println(x1[i]);`
	`}`

4. **Vòng lặp For Each (For Enhanced):**

+ for each == for enhanced
+ Sử dụng nhiều trong mảng dữ liệu Array
+ Công thức:

+ Khởi tạo dữ liệu mảng: <type>[ ] arrayName = {val1, val2, …};
+ Công thức chung:

```
for (<type> x : array) {
     # Thực hiện các khối lệnh lặp
}
```

+ dấu ":" có chức năng như "in" (trong)

VD:

Vòng lặp mảng	Vòng lặp chuỗi

`int[] mang = {1,2,3,4,5};`
`for (int x : mang) {`
`  System.out.println(x);`
`}`
	`String[] sts = {"Xin chào", "Trung tâm", "Java"};`
	`for (String s : sts) {`
	`  System.out.println(s);`
	`}`

5. **<u>Vòng lặp vô hạn:</u>**

+ Vòng lặp vĩnh cửu cần có điều kiện để check điều kiện vòng lặp và câu lệnh Break hay Continue

- Công thức:

```
while (true) {
        // Thực hiện câu lệnh lặp
        // Kiểm tra điều kiện nếu có
}
```

Bài 10: Một số câu lệnh quan trọng

1. **Câu lệnh Break:**
   + Tự động thoát ngay ra khỏi vòng lặp khi gặp “break;”
   + Chú ý:
      - Dùng nhiều trong vòng lặp để kiểm tra điều kiện
      - Giới hạn vòng lặp vô hạn
      - Có thể kết hợp với lệnh “switch case” để kiểm tra giá trị

   VD:

Vòng lặp for

```
for (int k = 0; k < 10; k++) {
    System.out.println(k);

    // Cách 1: Cách cơ bản
    if (k == 5) {
       break;
    }

    // Cách 2: Cách rút gọn
    if (k == 5)
       break;
}
```

Kiểm tra switch

```
int x = 1;

// Sử dụng switch case để kiểm tra
switch(x) {
     case 1: System.out.println(x);
     break;

     …
}
```

   ⇨ Tại câu lệnh break có thể không cần dấu ngoặc nhọn { }

2. **Câu lệnh Continue:**
   + Tiếp tục thực hiện chương trình khi gặp “continue;”
   + Bỏ qua các câu lệnh phía bên dưới “continue;” trong vòng lặp
   + Dùng để kết thúc sớm 1 lần lặp của vòng lặp

   VD:

CÁCH CƠ BẢN

```
for (int i = 0; i < 10; i++) {
    if (i < 5) {
        continue;
        // Tiếp tục vòng lặp khi đến continue;
        // Bỏ qua tất cả i < 5 đi và chạy tiếp
các giá trị sau
```

CÁCH RÚT GỌN

```
for (int i = 0; i < 10; i++) {
    if (i < 5)
        continue;
        // Tiếp tục vòng lặp khi đến continue;
    System.out.println(i);
}
```

```
    }
    System.out.println(i);
}
```

⇨ Tại câu lệnh continue có thể không cần dấu ngoặc nhọn { }

Bài 11: Một số thuật toán trong mảng

1. **Gán giá trị trong mảng dữ liệu:**

```
// Khởi tạo mảng ban đầu
int[ ] mangArray = new int[lengthArray];
Scanner scan = new Scanner(System.in);
for (int i = 0; i < mangArray.length; i++) {
        // Gán các giá trị của từng phần tử trong mảng
        mangArray[i] = scan.nextInt();
}

// Hiển thị ra màn hình dữ liệu mảng
for (int x : mangArray) {
        System.out.println(x);
}
```

2. **Tìm kiếm phần tử trong mảng:**

```
int k = scan.nextInt();
for (int i = 0; i < mang.length; i++) {
    if (k == mang[i]) {
        System.out.println("Phần tử k ở vị trí thứ " + i)
    }
}
```

3. **Thuật toán sắp xếp:**

```
for (int i = 0; i < mang.length; i++) {
    int temp = mang[i];
    for (int j = i + 1; j < mang.length; j++) {
        if (temp > mang[j]) {
            mang[i] = mang[j];
            mang[j] = temp;
            temp = mang[i];
        }
    }
}
```

// Hiện thị ra màn hình dữ liệu mảng đã sắp xếp

- Hoán đổi vị trí các phần tử cạnh nhau và lặp lại thao tác n lần (với n là số phần tử trong mảng)

4. **Thay đổi các phần tử trong mảng:**

+ Thay đổi trực tiếp giá trị ban đầu của phần tử đó:

VD:

```
+ Giá trị ban đầu của phần tử là: arr[i] = a
+ Thay đổi giá trị: a[i] = b    # Giá trị mới của phần tử thứ i trong mảng
```

5. **Xóa và chèn các phần tử trong mảng:**
    - Bản chất là tạo một mảng khác có kích cỡ lớn hơn (chèn) hoặc nhỏ hơn (xóa)

6. **Các phương thức với mảng:**
    - Arrays.copyOf(arr, <NewLength>) → Sao chép mảng mới

Bài 12: Xử lý chuỗi ký tự

1. **Cú pháp chuỗi ký tự:**

+ String bản chất giống như mảng ký tự, ta cũng có thể thao tác trong chuỗi giống thao tác với mảng. Tuy nhiên sẽ không sử dụng [ ] như mảng mà ta sẽ dùng phương thức: “charAt()”

- Cú pháp charAt():
  <chuỗi>.charAt(<index>);
  - <chuỗi> : string muốn truy xuất
  - <index> : chỉ số của ký tự trong mảng chuỗi

2. **Xử lý nhiều chuỗi với String Builder và StringBuffer:**

- StringBuilder với dữ liệu đơn luồng (ưu tiên tốt hơn)
  - StringBuilder builder = new StringBuilder();

- StringBuffer với dữ liệu đa luồng (tốc độ chậm hơn)
  - StringBuffer buffer = new StringBuffer();

=> Có mọi phương thức như mảng: str.append(); str.length(), str.reverse(), str.toString(), ...

3. **Một số hàm phương thức với chuỗi:**

a. Phương thức length:

- Phương thức trả về độ dài một chuỗi: ***“<chuỗi>”.length()”;***

```
Ví dụ:

String s = “Code”;
System.out.print(s.length());

→ KQ: 4
```

b. Phương thức replace:

- Thay thế các chuỗi hoặc ký tự được tìm thành chuỗi hoặc ký tự khác
- Công thức phương thức: ***“<chuỗi>”.replace(‘stringOld’, ‘stringNew’);***

```
Ví dụ:

System.out.println("3be".replace('3', 'e'));
System.out.println("Blackcat".replace("Black", "White"));

→ ebe
→ Whitecat
```

c. Phương thức toUpperCase và toLowerCase:

- Đây là hai phương thức để chuyển các ký tự của một xâu từ in thường về in HOA và ngược lại
- Công thức:
  - In HOA: “<chuỗi>”.toUpperCase();
  - In thường: “<chuỗi>”.toLowerCase();

```
Ví dụ:
String s = "qUanGTunG";
System.out.println(s.toUpperCase());
System.out.println(s.toLowerCase());

→ QUANGTUNG
→ quangtung
```

d. Phương thức indexOf:

- Phương thức này trả về vị trí xuất hiện đầu tiên của một String trong String khác, nếu không tìm thấy thì trả về -1
- Công thức: ***“<OldString>”.indexOf(“NewString”)***

```
Ví dụ:

String s = "QuangTung";
System.out.println(s.indexOf("Tung"));
System.out.println(s.indexOf("black"));

→ 5
→ -1
```

e. Phương thức startsWith và endsWith:

- Phương thức này dùng để kiểm tra một xâu có bắt đầu hoặc kết thúc bằng một xâu khác không
- Công thức:
  - ***“<Old String>”.startsWith(“NewString”);***
  - ***“<Old String>”.endsWith(“NewString”);***

```
public class Main {
    public static void main(String[] args) {
        String name = "Quang Tung";
        System.out.println(name.startsWith("Quang"));
        System.out.println(name.startsWith("abc"));
        System.out.println(name.endsWith("Tung"));
```

```
        System.out.println(name.endsWith("z"));
    }
}

→ true, false, true, false
```

f. Phương thức split:

- Phương thức tách một xâu ra thành mảng các chuỗi dựa trên một xâu cho trước
- Công thức: ***<type>[ ] NewArrayString = OldString.split("<khoảng trắng>");***

```
Ví dụ:

String s = "Welcome to Java";
String[] words = s.split(" ");
for (String word : words) {
        System.out.println(word);
}

→ Welcome
→ to
→ Java
```

g. So sánh hai chuỗi String với nhau:

- Công thức 1: ***String1.equalsIgnoreCase(String2)***
- Công thức 2: ***String1.equals(String2)***
- Công thức 3: s1.compareTo(s2)
    - Nếu len1 = len2 => kiểm tra từng char và trả về khoảng cách giữa 2 char đó
    - Nếu len != len2 => trả về len1 - len2

Bảng các phương thức trong chuỗi

Phương thức	Mô tả
char charAt(int index)	Trả về giá trị char cho chỉ số cụ thể.
int length()	Trả về độ dài chuỗi.
static String format(String format, Object... args)	Trả về chuỗi được format.
static String format(Locale l, String format, Object... args)	Trả về chuỗi được format theo vùng miền(quốc gia).
String substring(int beginIndex)	Trả về chuỗi con bắt đầu từ chỉ số index.
String substring(int beginIndex, int endIndex)	Trả về chuỗi con từ chỉ số bắt đầu đến chỉ số kết thúc.
boolean contains(CharSequence s)	Kiểm tra chuỗi ban đầu có chứa chuỗi s không, kết quả trả về là giá trị boolean.
static String join(CharSequence delimiter, CharSequence... elements)	Trả về chuỗi được nối từ nhiều chuỗi.
static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	Trả về chuỗi được nối từ nhiều chuỗi.
boolean equals(Object another)	kiểm tra sự tương đương của chuỗi với đối tượng.
boolean isEmpty()	Kiểm tra chuỗi rỗng.
String concat(String str)	Nối chuỗi cụ thể.
String replace(char old, char new)	Thay thế tất cả giá trị char cụ thể bằng một giá trị char mới.
String replace(CharSequence old, CharSequence new)	Thay thế tất cả các chuỗi bằng một chuỗi mới.
static String equalsIgnoreCase(String another)	So sanh chuỗi, không phân biệt chữ hoa hay chữ thường.
String[] split(String regex)	Trả về mảng các chuỗi được tách ra theo giá trị regex.
String[] split(String regex, int limit)	Trả về mảng các chuỗi được tách ra theo giá trị regex và có giới hạn.
String intern()	Trả về chuỗi interned.
int indexOf(int ch)	Trả về vị trí của ký tự ch cụ thể.
int indexOf(int ch, int fromIndex)	Trả về vị trí của ký tự ch tính từ từ vị trí fromIndex.
int indexOf(String substring)	Trả về vị trí của chuỗi con substring.
int indexOf(String substring, int fromIndex)	Trả về chuỗi con bắt đầu từ vị trí substring đến vị trí fromIndex.
String toLowerCase()	Trả về chuỗi chữ thường.
String toLowerCase(Locale l)	Trả về chuỗi chữ thường bằng việc sử dụng locale cụ thể.
String toUpperCase()	Trả về chuỗi chữ hoa.
String toUpperCase(Locale l)	Trả về chuỗi chữ hoa bằng việc sử dụng locale cụ thể.
String trim()	Xóa khoảng trắng ở đầu và cuối của chuỗi.
static String valueOf(int value)	Chuyển đổi giá trị kiểu dữ liệu đã cho thành chuỗi.

Bài 13: Lập trình hướng đối tượng

Nền tảng OOP = Lớp (Class) + Đối tượng (Object)

1. **Khởi tạo các thuộc tính:**
   - Các thuộc tính của đối tượng (tên, tuổi, địa chỉ, …) được khai báo ở ngoài hàm Main() mặc định
   - Cấu trúc như tạo biến dữ liệu: ***<type> Name;***
     VD:

```
public class B01_ClassObject {
    // Các thuộc tính của con người (biến đối tượng)
    String ten;
    String diaChi;
    int tuoi;

    // Hàm main
    public static void main(String[] args) {
        // Tạo đối tượng trong Java

    }
}
```

2. **Khởi tạo các hàm hành vi:**
   - Gọi hàm ở phía ngoài hàm main(), tức là tạo các hàm con nhỏ thể hiện hành vi của đối tượng (HÀM ĐỐI TƯỢNG)
   - Cấu trúc:

```
public void HoatDongHV(<type> TenBien1, …) {
        // Các hoạt động của đối tượng;
}
```

3. **Đối tượng chính trong hàm Main():**
   - Cách khởi tạo đối tượng trong hàm Main():
     - new ClassName(); // Khởi tạo đối tượng mới
     - ClassName <NewObject1> = new ClassName(); // Khởi tạo đối tượng 1
     - ClassName <NewObject2> = new ClassName(); // Khởi tạo đối tượng 2
     - …
   - Kết hợp đối tượng với hành vi:
     - NewObject.HoatDongHV(TenBien1, …)

VD:

```java
package JavaProgramming;
public class B01_ClassObject {
    // Các thuộc tính của con người (biến đối tượng – biến instance)
    String ten;
    String diaChi;
    int tuoi;

    // Hành vi của con người (hàm đối tượng)
    public void diLai(String ten) {
        System.out.println(ten + " đi lại");
    }

    // Hàm main
    public static void main(String[] args) {
        // Tạo đối tượng trong Java
        new B01_ClassObject();          // Đối tượng
        B01_ClassObject AnhA = new B01_ClassObject();   // Đối tượng 1
        B01_ClassObject AnhB = new B01_ClassObject();   // Đối tượng 2

        // Hành vi "đi lại" của từng đối tượng
        AnhA.diLai("Anh A");
        AnhB.diLai("Anh B");
    }
}
```

- Tạo đối tượng nâng cao:
  ObjCha nameBien = new ObjCon();

4. Biến và hàm Instance:

a. Thao tác với file đối tượng:

- Tạo file Đối tượng riêng khác file hàm main chính
- Hàm đối tượng chính thường ở trong file đối tượng Class
- Công thức:

```java
public class ObjectName {
    // Biến instance (biến đối tượng) và khai báo ngay dưới class
    <type> Tên_Biến;

    // Tạo hàm Hành Vi cho đối tượng
    public void setTen(String ten) {
        // this.ten là lấy từ biến instance
```

```
        this.ten = ten;
    }

    // Ứng dụng hàm Set - Get trong Java
    public String getTen() {
        return ten;
    }

    // Hàm đối tượng
    public void anUong() {
        System.out.println("Ăn uống");
    }
}
```

- Hàm SET đối tượng:
  - Lưu và đưa về biến đối tượng
  - Cấu trúc:

```
public void setName(<type> Bien1, ...) {
        // this.ten là lấy từ biến instance
        this.ten = ten;
}
```

- Hàm GET đối tượng:
  - Trả lại biến dữ liệu ở trên (nếu muốn đưa kết quả ra màn hình)
  - Trả lại biến đối tượng đó ra bên ngoài
  - Cấu trúc:

```
public <type> getName() {
        return ten;
}
```

b. Thao tác với file chính chạy hàm Main():

- Tạo đối tượng mới từ file Class ObjectName
- Gọi hàm đối tượng (hàm hành vi) của đối tượng đó

c. Constructor:

- Constructor nằm giữa biến instance và hàm instance
- Gộp các biến đối tượng lại chung với nhau, tối ưu hóa cách viết của setter
- Kết hợp hàm Getter để hiển thị đối tượng ra ngoài
- Công thức:

```
public NameObject(<type1> name1, ...) {
        this.name1 = name1;
        this.name2 = name2;
```

```
        ...
}
```

d) Phương thức hiển thị dữ liệu đối tượng:

```
public String toString() {
    // Các khối lệnh bên trong
}
```

- Đưa kiểu dữ liệu đối tượng về kiểu String để hiển thị được dữ liệu ra bên ngoài

5. Biến và hàm Class Static:

- Biến static được dùng chung cho mọi đối tượng
- Có thể bị thay đổi tùy vào đối tượng
- Công thức:
  - ***static <type> Tên_Biến = <value>;***
  - ***public statics <type> Tên_Biến;***

6. So sánh giữa các loại biến và hàm:

Biến Local	Biến và hàm Instance	Biến và hàm Static

+ Biến được khai báo trong hàm và chỉ tồn tại trong hàm đó
+ Thường được khai báo trong hàm instance
 	
VD biến local và hàm instance:
`// Hàm instance`
`public void xemPhim() {`
`  // Biến Local`
`  String xem = "Xin chào";`
`  System.out.println(xem);`
`}`
	+ Biến instance là biến đối tượng và được khai báo ngay phía dưới class
	
	VD biến instance:
	`public String tenPhim;`
		+ Biến static được dùng chung cho mọi đối tượng và sẽ thay đổi tùy vào đối tượng
		
		VD biến và hàm static:
		`// Biến static`
		`public static String giaPhim;`
		`// Hàm static`
		`public static void giaPhim() {`
		`  System.out.println("");`
		`}`

- So sánh static và final:
  - Nếu dùng static thì không cần khởi tạo Object cho lớp đó
    - public static void nameFunction() {}
  - Dùng final nếu không muốn lớp con ghi đè phương thức từ lớp cha
    - public final void nameFunction() {}

7. Quản lý truy cập và bảo mật dữ liệu:

BẢNG SO SÁNH ĐỘ BẢO MẬT				
Có thể gọi	Public	Protected	Default	Private
cùng class	Yes	Yes	Yes	Yes
cùng package	Yes	Yes	Yes	No
subclass cùng package	Yes	Yes	Yes	No
subclass khác package	Yes	Yes	No	No
class khác package	Yes	No	No	No

Lưu ý:

- Tạo đối tượng nên dùng private
- Tính trừu tượng kế thừa nên dùng protected

+ Biến instance hay để loại private
+ Hàm Set - Get dùng public (hạn chế truy cập trực tiếp vào biến)

Ví dụ:

- Khai báo các biến:

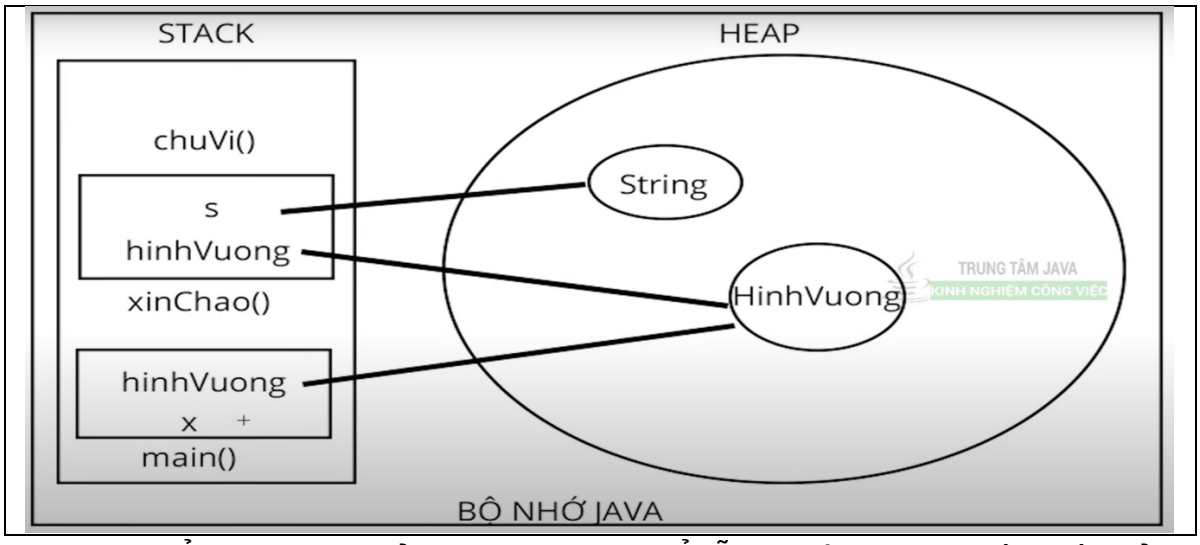
```
// Khai báo các biến có tính năng khác nhau
String ten;
private int tuoi;       // Chỉ truy cập trong cùng class
public String diaChi;
protected String danToc;
```

- Khai báo với class:

```
// Khai báo class con khác trong class chính
private class User {}
```

8. Bộ nhớ Stack và Heap:

Bộ nhớ Stack	Bộ nhớ Heap
+ Dung lượng nhỏ + Lưu các biến cục bộ, hàm tham chiếu	+ Dung lượng lớn hơn + Dùng để lưu các biến đối tượng + Hay có biến rác và được giải phóng bộ nhớ



- Kiểu dữ liệu cơ bản thường dùng 0 để rỗng, không mang giá trị nào cả
- Kiểu dữ liệu đối tượng dùng null để chỉ đến những biến không mang giá trị nào cả (chưa trỏ đến đối tượng nào)

9. Mảng đối tượng:

- Khá giống với khai báo mảng cơ bản (coi đối tượng là <type>)
- ClassName có n thuộc tính thì mỗi ClassName cũng có tương ứng n biến
- Công thức:

  ClassName[ ] NewNameArray = new ClassName[<số lượng ĐT>];
- Xóa bộ nhớ đệm: ***scan.nextLine();***
- Truy xuất mảng dữ liệu:

  NewNameArray[i] = new ClassName(arg1, arg2, …)

  - ClassName khi đó đóng vai trò như <type>
  - NewNameArray đóng vai trò là tên mảng dữ liệu

10. Tính kế thừa:

- Kế thừa là khả năng thừa hưởng những biến và hàm của class khác
- Có khả năng rút ngắn thời gian, viết code ngăn hơn
- Chỉ được kế thừa duy nhất từ 1 class
- Các class có thể kế thừa liên tiếp nhau
  - Class kế thừa = subclass (class con)
  - Class được kế thừa = superclass (class cha)

⇨ Superclass có thể có nhiều Subclass, nhưng Subclass chỉ có duy nhất 1 Superclass

- Công thức: (từ khóa Extends)

```
public class NewClass extends OldClass {
```

```
        // NewClass được kế thừa từ OldClass
        // Thực hiện tiếp các chức năng mới trong NewClass
}
```

- So sánh từ khóa “this” và từ khóa “super”:

this.name	super.name
Hướng tới đối tượng ngay trong class đó	Chỉ hoạt động trong class con (class kế thừa) và hướng đến đối tượng cha (class được kế thừa) # Truy xuất ngược lại tới class cha

VD:

super.tenHam(); ***// Chỉ lại tới hàm của class cha***
super.bien; ***// Chỉ lại tới biến của class cha***

- So sánh giữa “Is A” và “Has A”:

Is – A	Has - A
+ Ứng dụng mạnh trong kế thừa đối tượng + Đối tượng B kế thừa từ đối tượng A tức là đối tượng B cũng là A nhưng ngược lại không đúng	+ Mỗi đối tượng có một địa chỉ truy cập riêng + Có tính chất biến đối tượng đó thành <type> <u>MÔ TẢ:</u> + Class B này đều có một số đặc tính riêng từ Class A khác + Đối tượng A đóng vai trò như <type> trong class B

- So sánh Override và Overload:

Overload	Override

+ Là những phương thức có tên hàm giống nhau nhưng đối số truyền vào khác nhau
+ Chú ý ở kiểu dữ liệu truyền vào
 	
+ Cấu trúc:
public <type1> tenHam(<ty> arg1, arg2, …) {
    // Các phương thức
}
	+ Viết lại một hàm của class cha trong class con nhưng có thể ghi đè lên hàm cha (viết lại phương thức bên trong của hàm cha)
	
	+ Cấu trúc:
	@Override
	public <typeOldClass> nameOldClass(<type> nameBien) {
	    // Các phương thức
	}

VD:
`// Hàm tính tổng hai số (số nguyên)`
`public int tongHaiSo(int a, int b) {`
`    return a + b;`
`}`
	VD:
	`@Override`
	`public float tongHaiSo(float a) {`
	`// TODO Auto-generated method stub`
	`    return a + 20;`
	`// return super.tongHaiSo(a);`

```java
// Hàm tính tổng hai số (số thực)
public float tongHaiSo(float a, float b) {
    return a + b;
}
```

11. Tính đa hình:

- Từ class cha hướng chỉ đến class con
- Một đối tượng con có 1 class cha (kiểu dữ liệu class cha được áp cho class con)
- Công thức:

<OldClass> BienName = new <NewClass>

+ Ta có thể khai báo đối tượng này bằng kiểu đối tượng khác
+ Tính đa hình sẽ luôn lấy theo dữ liệu của <NewClass>
+ <NewClass> và <OldClass> có thể giống nhau

VD:

```java
// Lấy theo dữ liệu của hàm trong Employmental
Employmental emp = new Employmental();

// Lấy theo dữ liệu của hàm trong Employmental
Personalize p = new Employmental();
```

12. Kiểu đối tượng Class Object:

- Class Object là cha của mọi class
- Class Object là kiểu dữ liệu lớn nhất bao trùm được chỉ đến tất cả mọi đối tượng
- Cấu trúc:

Object tenBien = new <NameClass>;

- Ép kiểu dữ liệu đối tượng:
  - Ép xuống từ kiểu dữ liệu cha về kiểu dữ liệu con (ép từ kiểu dữ liệu lớn hơn về kiểu dữ liệu nhỏ hơn)
    - 1. Tạo cha và ép xuống con (cẩn thận)
    - 2. Tạo con và ép về cha là được (an toàn)
  - Cấu trúc:
    <OldClass> tenBien = new <NewClass>;
    - <newClass> BienName = (<type Object>) tenBien;
    - Viết tắt: ((<type Object>) tenBien).tenHam;

13. Toán tử Instanceof:

- Trả về gia trị True/False
- Check kiểu dữ liệu đối tượng của biến tham chiếu (biến đối tượng)

- Kiểm tra xem biến này có phải con của kiểu dữ liệu này không
- Cấu trúc: ***tenBien instanceof DoiTuong***

14. Tính đóng gói:

- Được ứng dụng và sử dụng trong quản lý truy cập để khai báo các hàm, các biến (Public, Protected, Default, Private)
- Tính chất kế thừa nên dùng protected

15. Abstract Class:

- Chuyên sinh ra để làm cha (không thể tạo thêm đối tượng mới này khi đó là Abstract Class)
- Chuyên áp dụng cho kế thừa (làm class cha)
- Không thể lưu đối tượng từ Abstract Class
- Công thức:

```
1. Tại đối tượng Abstract Class:
public abstract class <OldClass> {
     // Các thao tác bên trong
}

2. Tại đối tượng kế thừa:
public class <NewClass> extends <OldClass> {
    // Các thao tác bên trong
}

3. Hàm main chính:
<NewClass> BienName = new <NewClass>;
<OldClass> BienName = new <NewClass>;
```

- Phương thức của Abstract Class:
  - Bắt buộc các class con phải Override lại hàm Abstract của cha
  - Các hàm Abstract chỉ tồn tại trong class Abstract
  - Class Abstract có thể chứa hoặc không chứa các hàm Abstract
  - Các class Abstract có thể kế thừa lẫn nhau
  - Công thức:

1. Hàm Abstract:
public abstract <type> tenHam();

2. Class Abstract:
publict abstract class <NameClass> {
}

3. Kế thừa Abstract Class:

```
publict abstract class <OldClass> extends <NameClass> {
}
```

16. Interfaces:

- Tổng quan Interfaces:
  - Interfaces là cấp độ cao hơn của Abstract class (class cha của mọi class)
  - Trong interfaces chỉ chứa các hàm Abstract và hằng số
  - Không thể khai báo hàm bình thường hay biến trong đó
  - Mọi hàm đều để chế độ public và phải rỗng ở trong đó (khi override thì viết lệnh vào trong đó) → Viết tạm hàm rỗng để về sau thêm sửa sau
- Cấu trúc:

1. Class interfaces:

```
public interface <NameClass> {
    // Khai báo hằng số
    public static int HANG_SO = <value>;

    // Các hàm Abstract
    public abstract void func1();
    public void func2();      // Ngầm là public abstract void func2();
    void func3();     // Ngầm là public abstract void func3();
}
```

2. Kế thừa interfaces:

+ Thay từ khóa “extends” thành “implements” nếu hàm kế thừa bình thường
+ Override tất cả các hàm trong class cha trong class con

```
public class <NewClass> implements <OldClass> {
    @Override
    public void Ham() {
        // TODO Auto-generated method stub
    }
}
```

+ Các class Interfaces có thể kế thừa lẫn nhau bằng từ khóa “extends”
+ Class Interfaces có thể kề thừa cùng lúc nhiều class cha khác bằng dấu phẩy
+ Tính đa hình được ứng dụng nhiều trong Interfaces

```
public interface <NameClass> extends <ClassName1>, <ClassName2> {
    // Các thao tác bên trong
}
```

17. Wrapper Class và Autoboxing:

- ***Wrapper Class:***
  + Là các class tương ứng với kiểu dữ liệu cơ bản

Kiểu đối tượng (<Type>)	Kiểu cơ bản (<type>)
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Boolean	boolean
Character	char

+ Khai báo biến kiểu Wrapper – kiểu đối tượng tham chiếu:

<Type> BienName = new <Type>(value);

+ Dùng Wrapper class để convert kiểu dữ liệu:

<type> tenBien = <Type>.valueOf(giaTri);
<type> tenBien = <Type>.parseInt(giaTri);
<type> tenBien = <Type>.MIN_VALUE → giá trị nhỏ nhất
<type> tenBien = <Type>.MAX_VALUE → giá trị lớn nhất

VD:

```
// Convert kiểu dữ liệu → thực hiện được chức năng cơ bản
int z1 = Integer.valueOf("45");
int z2 = Integer.parseInt("45");
System.out.println(z1 + z2);
```

- ***Autoboxing:***
  - Boxing: tự chuyển từ kiểu cơ bản thành kiểu đối tượng
  - Unboxing: chuyển từ đối tượng về cơ bản

⇨ Có thể so sánh giữa các loại với nhau (ĐT vs CB, CN vs CB) nhưng đặc biệt không thể so sánh được các đối tượng độc lập được khởi tạo (ĐT vs ĐT)

Unboxing	Boxing

int x = 10;
Integer y = new Integer(10);
int z = x + y;
=> chuyển đổi Integer về int
	Integer k = 10;
	k = new Integer(10);

- ***Hàm điều kiện thay giá trị: (phương thức đối tượng)***
  - Tương tự với câu lệnh hàm so sánh mặc định Equals():

```
NameValueInput.equals(Value)
```

```
String1.equalsIgnoreCase(String2)
```
⇨ Gán giá trị đầu vào NameValueInput với giá trị Value

- Hàm so sánh hai chuỗi “String”
- Phương thức so sánh giữa các đối tượng với nhau: ***DT1.equals(DT2)***

18. Outer Class:

- Outer Class là những Class dạng default được khai báo cùng lúc bên ngoài của file đó
- Trong mỗi file phải luôn có ít nhất một public class trùng với tên file
- Các class khác có thể kế thừa Outer class (nhưng thông thường) nếu chúng ở trong cùng Package
- Cấu trúc:

```
package <NamePackage>;

public class <NameFile> {
    // Thực hiện các chức năng chính bên trong
}

class <NameOuterClass> {
   // Thực hiện các chức năng bên trong
}
```

VD:

```
package JavaProgramming;

public class B25_OuterClass {
    public void xinChao() {
        Kid kid = new Kid();
        kid.hello();
    }
}

// Class dạng default được khai báo cùng lúc bên ngoài file đó
/* Lưu ý:
 * + Trong mỗi file có ít nhất một public class trùng với tên file
 * + Các class khác có thể kế thừa Outer class nếu cùng Package
 */
class Kid {
    private int tuoi;
    public void hello() {
```

```
        System.out.println("Xin chào");
    }
}
```

19. Inner Static Class:

- Là các class static được khai báo bên trong class đối tượng
- Có thể tùy chọn quản lý truy cập khác nhau
- Có thể truy cập nếu khác package tùy thuộc vào quản lý truy cập
- Cấu trúc:

```
package <NamePackage>;

public class <NameClassFile> {
    // Khởi tạo inner static class
    public static class <NameClassStatic> {
        // Khai báo các chức năng như bình thường
    }
}
```

- Đối với hàm Main mặc định:
    - Các gọi thông qua Static: (gọi đối tượng con trong đối tượng lớn)

```
<NameFile>.<NameClassStatic> tenBien = new <NameFile>.<NameClassStatic>();
```

- Đối với Lớp kế thừa:

```
public class <NewClassName> extends <NameFile>.<NameClassStatic> {
    // Thực hiện các thao tác bên trong lớp kế thừa
}
```

20. Non Static Inner Class:

- Khởi tạo Inner Class trong Class File đối tượng:

```
public class <NameFile> {
        // Khởi tạo Inner Class
        "public" class <NameObject> {
                // Thực hiện các chức năng cơ bản trong Inner Class
        }
}
```

- Trong hàm main chạy chính trong cùng class File:
    - Tạo đối tượng file: ***<NameFile> NameBienFile = new <NameFile>();***
    - Từ đối tượng file tạo đối tượng trong Inner Class:

<NameFile>.<NameObject> TenBien = NameBienFile.new <NameObject>;

- Đối với lớp kế thừa:
  - Nếu trong cùng class File có thể kế thừa luôn Inner Class
  - Nếu khác class File không thể kế thừa được Inner Class
  - Các class khác class File vẫn có thể chạy hàm main của class File chứa các chức năng đối tượng Inner Class đó

- Phương thức Local Inner Class:
  - Khai báo Class Inner bên trong phương thức hàm với dạng Default
  - Khi tạo đối tượng Inner Class thì chỉ có thể truy xuất được hàm, biến bên trong Inner Class đó

VD:

```
// Khai báo Class Inner bên trong phương thức hàm
public class Employee {
    // Tạo hàm phương thức
    public void luong() {
        // Inner Class dạng default
        class Worker {
            private String ten;
            public void luong() {
                System.out.println("Lương công nhân");
            }
        }

        // Tạo đối tượng
        Worker w = new Worker();
        w.luong();  // Hàm lương trong Worker
    }

    public static void main(String[] args) {
        Employee em = new Employee();
        em.luong();   // Hàm lương của Employee
    }
}
```

21. Anonymous Inner Class:

- Được áp dụng đối với interface để khởi tạo hàm rỗng hoặc đối với Abstract Class để tạo Override kế thừa
- Bản chất Anonymous Class là những class không được đặt tên:

InterfaceClass <tenBien> = new InterfaceClass() {
// Override lại các hàm trong đối tượng Interface

```
};
```

VD:

```java
// Áp dụng với file class interface ServicePerson.java
public class B29_AnonymousInnerClass {
    public static void main(String[] args) {
        // Tính đa hình của interface
        ServicePerson p = new ServicePerson() {

            // Phải override lại hàm
            @Override
            public void hello() {
                // TODO Auto-generated method stub
                System.out.println("Hello");
            }
        };
        p.hello();

        // Gọi Abstract Class
        XinChao x = new XinChao() {
            // Phải Override
            @Override
            public void xinChao() {
                // TODO Auto-generated method stub
                super.xinChao();
            }
        };
        x.xinChao();
    }
}

// Khởi tạo Abstract Class
abstract class XinChao {
    public void xinChao() {
        System.out.println("Xin chào");
    }
}
```

22. Khởi tạo Enum:

- Đổi loại class thành enum – hằng số cho class:
    - Trong class enum luôn chỉ sử dụng private trong Constructor
    - Dùng dấu phẩy “,” để cách ra tạo các hàm hằng

- Cách thức 1: (Khai báo biến hằng Enum)

```
enum ClassName {
    // Khai báo các hằng số mặc định
    VAL1, VAL2, ...;
}
```

- Cách thức 2: (Khai báo giá trị cho Enum khi có Constructor)

```
enum ClassName {
    VAL1("cafe"), VAL2("trà"), VAL3("coca");       // Luôn là hằng số
    private <type> tenBien;

    // Khởi tạo Constructor
    private MyEnum(String tenBien) {
        this.tenBien = tenBien;
    }

    // Khởi tạo Getter và Setter
    public String getTenBien() {
        return tenBien;
    }
    public void setTenBien(String tenBien) {
        this.tenBien = tenBien;
    }
}
```

- Cách thức 3: (Trừu tượng hóa Enum Abstract)

```
enum ClassName {
        // Khai báo các hằng số
        VAL1("something"){
           @Override
           // Abstract Function
        };
        abstract function();
}
```

- Cách gọi phương thức/thuộc tính trong main:
  - Công thức 1: ***MyEnum.Fuct.getTen();***
  - Công thức 2: ***MyEnum.VAL;***

4 tính chất cơ bản của OOP

Tính đóng gói (encapsulation) --> tăng tính bảo mật object
Tính kế thừa (inheritance) --> con kế thừa lớp cha bằng 'extends'
Tính trừu tượng (abstraction) --> abstract class, interface
Tính đa hình (polymorphism) --> overloading, overriding

Bài 14: Thư viện Toán học và Thời gian

1. **Thư viện Toán học cơ bản:**

   + Không cần import thư viện Math do Java đã tích hợp sẵn tính năng thư viện Toán học

Công thức	Mô tả
Math.abs(<value>);	Hàm trả giá trị tuyệt đối của số
Math.PI;	Hàm tra về giá trị số PI
Math.sin(<value>);	Hàm trả về giá trị lượng giác (tính theo radian)
Math.cos(<value>);	
Math.tan(<value>);	
Math.pow(a, b);	Hàm lũy thừa a mũ b (a^b) với a là cơ số, b là số mũ
Math.sqrt(<value>);	Hàm trả về căn bậc hai của một số
Math.random();	Giá trị ngẫu nhiên từ 0 – 1
Math.ceil(<value>);	Làm tròn số lên trên
Math.floor(<value>);	Làm tròn số xuống dưới
Math.max(a, b);	Tìm giá trị lớn nhất trong hai số
Math.min(a, b);	Tìm giá trị nhỏ nhất trong hai số
Math.round(a*100)/100	Làm tròn a hai chữ số thập phân

2. **Thư viện xử lý dữ liệu số:**

   + Dùng để định dạng làm tròn số: DecimalFormat

   - import java.text.DecimalFormat;

```
DecimalFormat dcf = new DecimalFormat("#.###");
System.out.println(dcf.format(số muốn làm tròn));
```

     - # là ẩn số định dạng (tùy chỉnh)

3. **Thư viện Thời gian:**

   + Import thư viện Toán học: Date hoặc Calendar

   - import java.util.Date; → khó tính toán
   - import java.util.Calendar;

a) Thư viện Date:

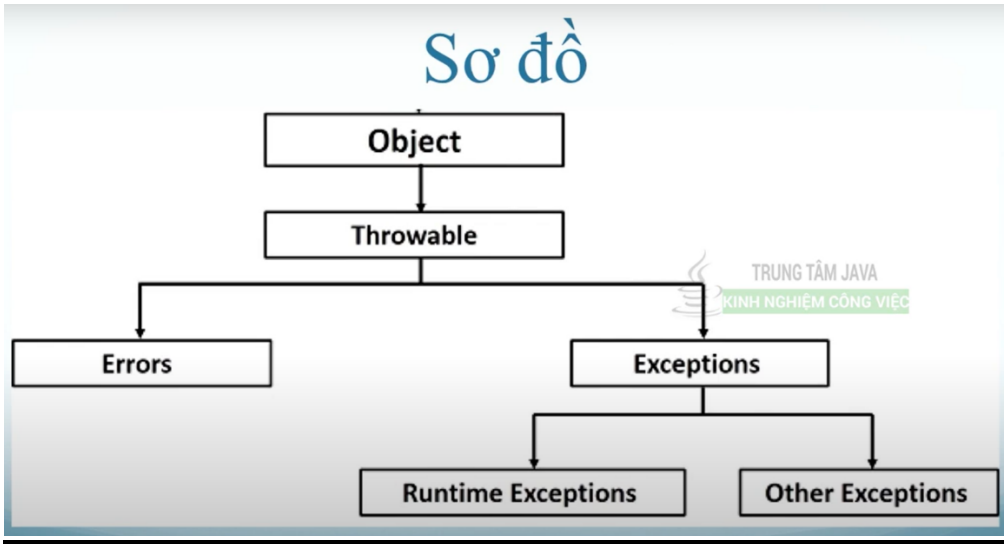
   - Hiển thị thời gian hiện tại: Date now = new Date();
   - So sánh giữa hai khoảng thời gian:
     + Ngang bằng: now1.equals(now2);
     + Trước: now1.before(now2);
     + Sau: now1.after(now2);
   - Đổi thời gian hiện tại: now.getTime();
   - Đổi thời gian mong muốn sang mili giây: now.setTime(<valueTime>);

b) Thư viện Calendar:

- Hiển thị thời gian hiện tại: Calendar cal = Calendar.getInstance();
- Hiển thị thời gian: cal.getTime();
- Thêm bớt thời gian: cal.add(Calendar.Thoigian, <thembot>);
    - Thoigian có thể là: HOUR, MINUTE, SECOND, YEAR, …
    - <thembot> là thêm hoặc bớt giá trị của Thoigian

Bài 15: Ngoại lệ Exception

1. **Tổng quan về Exception:**
    - Exception là một lỗi xảy ra trong quá trình chạy khiến cho chương trình bị dừng lại và cần tránh trong thực tế
    - Exception là cha của mọi class ngoại lệ
    - Các loại Exception:
        - Checked Exception: có thể biết trước ở compile time
        - Unchecked Exception: Không đoán trước được do code kém
        - Errors: lỗi máy ảo, thiếu bộ nhớ không thể fix trong code



2. **Cấu trúc try … catch:**
    - Mục đích dùng để bắt lỗi từ chương trình
    - Cách thức hoạt động:
        - Sẽ kiểm tra điều kiện try trước
        - Nhận diện loại Exception rồi chạy vào catch tương ứng theo thứ tự
    - Cấu trúc cơ bản:

try … catch	try … catch … catch

try {
  // Khối lệnh thực thi điều kiện
} catch (Exception e) {
  // Nếu lỗi sẽ thực hiện câu lệnh này
 	
  // Cách 1:
	try {
	  // Khối lệnh thực thi điều kiện
	} catch (NameError <name>) {
	  // Nếu lỗi sẽ thực hiện câu lệnh này
	}
	} catch (Exception e) {

System.out.println("Lỗi Errors" + e);
 	
// Cách 2:
e.printStackTrace()
}
 	
=> e là thông tin cụ thể lỗi đó
	// Nếu lỗi sẽ thực hiện câu lệnh này
	}
	
	+ Luôn để NameError lên trước Exception

- Cấu trúc nâng cao:

Finally Exception

```
try {
    // Khối lệnh try
} catch (<NameError> e) {
    // Đối với nhiều Exception
} catch (Exception e) {
    // In ra thông tin lỗi
} finally {
    // Luôn được in ra sau khi chạy khối lệnh try ... catch
    // Luôn được thực thi dù có Exception không
    System.out.println("Finally");
```

=> Luôn thực thi kể cả có hay không có Exception

3. Throw và Throws Exception:

- Định nghĩa:
  - throw: Để ném ra một ngoại lệ cụ thể (tự thiết kế)
  - throws: Để thông báo một phương thức có thể xảy ra ngoại lệ
- throw/throws là cách đẩy ra các exception:
  - Nếu khởi tạo hàm dùng throws thì không cần try … catch và ngược lại
  - Hàm được throws có chức năng tạo ra tên cụ thể exception
  - Hàm cần phải throws trước rồi trong hàm main sử dụng tiếp try ... catch

⇨ Các loại Throw và Throws:

- Tạo hàm mới để bắt các ngoại lệ:
  - Cấu trúc:

```
public static void NameFunction() throws <NameError> {
      // Khối lệnh exception của hàm
      Luôn phải có try … catch trong hàm throws
      try {
          // Khối lệnh thực thi
      } catch (Exception e) {
          throw e;
      }
```

```
}
```

- Tạo đối tượng Exception mới khi đã biết ngoại lệ:
  - Cấu trúc:

```
public static void FunctionName() {
        // Dùng throw để tạo đối tượng exception mới
        throw new NameError();
}
```

- Tạo đối tượng Exception mới khi chưa biết ngoại lệ:
  - Cấu trúc:

```
public static void NameFunc throws <NameError> {
        throw new NameError();
    }
```

- Cấu trúc try … catch trong hàm main:

```
 try {
     tenHamException();
} catch (NameError e) {
     System.out.println("e");
}
```

4. **<u>Khởi tạo Exceptions bất kỳ:</u>**
   - Tạo Class kế thừa ngoại lệ Exception:

```
public class MyException extends Exception {
    private String error;

    // Tạo Constructor
    public MyException(String error) {
        super();
        this.error = error;
    }

    // Tạo hàm Setter và Getter
    public String getError() {
        return error;
    }
    public void setError(String error) {
        this.error = error;
}
```

   - Trong File Class chính:

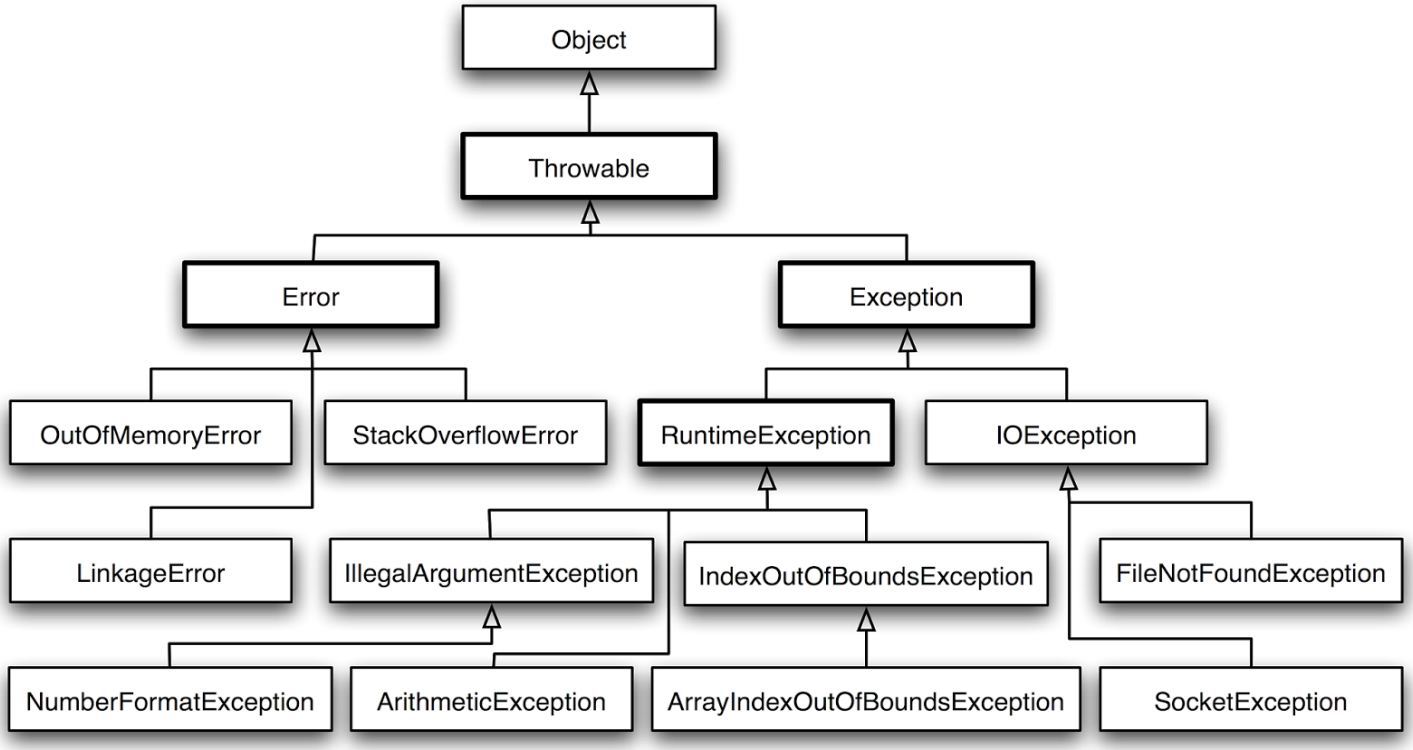
- Khởi tạo hàm ngoài hàm main:

```
public static void TenHam(<type> a, <type> b) throws MyException {
        try {
            // Thực hiện khối lệnh try
        } catch (Exception e) {
            // Khởi tạo đối tượng Exception
            throw new MyException("Lỗi Exception");
        }
}
```

- Trong hàm Main:

```
// Vẫn phải sử dụng try … catch
try {
     TenHam(a, b);
} catch (MyException e) {
     System.out.println(e.getError());
}
```

5. Sơ đồ Ngoại lệ Exception:



Bài 16: Dữ liệu đa luồng Thread

Tiến trình > Luồng

1. **Đồng bộ và bất đồng bộ: (SYNCHRONIZED & UNSYNCHRONIZED)**
   - Synchronized (đồng bộ) → xử lý tuần tự
   - Unsynchronized (bất đồng bộ) → tối đa hóa khả năng và tốc độ xử lý (thực hiện song song và giảm bớt thời gian xử lý)

2. **Dữ liệu đa luồng Thread:**
   - Trong cùng 1 thời điểm có thể xử lý nhiều công việc chạy song song với nhau
   - Các kết quả in ra khác nhau do các class đó được chạy song song với nhau
   - Có 2 cách tạo Thread: implements interface và extends Thread
   - Cho đối tượng chạy: ***<nameDT>.start();***

Implements Interface

```
public class MyThread1 implements Runnable {
    @Override
    public void run() {
        // Khối lệnh chạy song song dưới hàm run()
    }

}
```

Extends Thread

```
public class MyThread2 extends Thread {
    @Override
    public void run() {
        // super.run();
        // Khối lệnh chạy song song
    }
}
```

HÀM MAIN

```
MyThread1 my1 = new MyThread1();
Thread t = new Thread(my1);
t.start();
```

```
MyThread2 my2 = new MyThread2();
my2.start();
```

- So sánh thread.start() và thread.run():
  - start(): Khởi tạo 1 thread với path riêng rẽ. sau đó gọi phương thức run() và thực hiện trên thread đó.
  - run(): Bản chất nếu Thread được khởi tạo là 1 đối tượng Runnable, phương thức run() sẽ được gọi

- Thread luôn có trạng thái bị ngủ đông hoặc bị dừng một khoảng thời gian để chạy

3. Câu lệnh ngủ đông Sleep() trong Thread:

- Xử lý trong hàm run() được @Override
- Câu lệnh ngủ đông khoảng thời gian chạy sleep():
  - Class phải extends từ Thread
  - Trong run() có sử dụng câu lệnh kèm với try … catch đối với hàm sleep():
    - Cấu trúc: Thread.sleep(<miliS>)
    - <imliS> là thời gian nghỉ để chạy

VD:

```
public class B03_JoinVaDatTenThread extends Thread {
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.out.println(this.getName() + " " + i);
        }
    }
}
```

4. Cách đặt tên Thread:

- Khởi tạo đối tượng trong hàm main như thông thường
- Đặt tên đối tượng bằng cách: ***<NameDT>.setName("Tên Thread");***
- Hướng chỉ tới đối tượng trong hàm run() được @Override là:
  - ***C1: this.getName();***
  - **C2: Thread.currentThread().getName();**

- Cấu trúc:

```
ClassName <NameDT> = new ClassName();
<NameDT>.setName("Tên Thread");
```

5. Câu lệnh Join đa luồng chạy:

- Cho phép đối tượng chạy song song với nhau với điều kiện cho sẵn
- Giống với việc chạy đối tượng kia chạy trong khoảng thời gian nhất định rồi những đối tượng mới được chạy song song cùng tiếp
- Có phải kết hợp chạy cùng với try … catch
- Cấu trúc: <nameDT>.join(<miliS>);

- Nếu <miliS> rỗng thì sẽ chạy hệt <nameDT> rồi mới chạy tiếp những đối tượng sau
- Giả sử: <miliS> = k thì <nameDT> sẽ chạy trong k giây đầu tiên, sau đó chạy song song cùng các đối tượng còn lại khác

VD:

```
t1.start();
try {
    t1.join(1500);
} catch (InterruptedException e) {
    e.printStackTrace();
}
t2.start();
t3.start();
```

=> t1 chạy trong 1500mls đầu tiên, sau đó chạy song song tiếp với t2, t3

6. Thread Priority và Deamon Thread:

a) Thread Priority:

- Ưu tiên Thread nào chạy trước và chạy sau
- Mang tính hình thức ý nghĩa
- Không đảm bảo thứ tự chạy của Thread
- Công thức: ***<NameDT>.setPriority(<ValueTime>);***

b) Deamon Thread:

- Thread Service chạy ở Background
- Được chạy ngầm bên dưới để dọn rác
- Mục đích chính là kiểm tra xem đối tượng có phải Deamon Thread hay không?
- Công thức:
  - <tenDT>.setDeamon(true); → Đưa đối tượng về Deamon
  - <tenDT>.isDeamon(); → Kiểm tra xem có là Deamon không?

7. Synchronization Thread:

a) Phương thức hàm:

- Đồng bộ hóa các Thread để cùng truy xuất 1 hàm
- Chúng sẽ được truy xuất lần lượt, những Thread sau sẽ đợi Thread trước chạy
- Class file phải "implements Runable"
- Khai báo từ khóa "synchronizied" trước hàm cần đồng bộ hóa:

```
// Hàm Synchronized
public synchronized void tenHam (val) {
     // Thực hiện các chức năng
}
```

```
<ClassName> tenBien = new <ClassName>();

Thread t1 = new Thread(tenBien);
Thread t2 = new Thread(tenBien);
```

→ Cho phép tất cả các luồng sử dụng dữ liệu trong hàm bằng từ khóa "synchronized"

b) Đối tượng khóa chung:

- Tuần tự trong đa luồng, thực hiện xong thread1 thì mới chuyển sang thread2 lần lượt

```
private static final Object lock = new Object();  // Đối tượng khóa chung
synchronized (lock) {
    // Khối lệnh thực hiện ưu tiên của từng luồng
}
```

8. Deadlock Thread:

- Deadlock sinh ra khi các hàm đều chứa nhau, khi đó các đối tượng sẽ chạy vô hạn lần lượt và sẽ đột ngột dừng, báo lỗi

VD:

```
public class B06_DeadlockThread implements Runnable {
    // Thread t2 chiếm giữ test1
    public synchronized void test1() {
        System.out.println("Test 1");
        test2();
    }

    // Thread t1 chiếm giữ test2
    public synchronized void test2() {
        System.out.println("Test 2");
        test1();
    }


    @Override
    public void run() {
        test1();
    }

    // Hàm main
    public static void main(String[] args) {
        B06_DeadlockThread d = new B06_DeadlockThread();
        Thread t1 = new Thread(d);
        Thread t2 = new Thread(d);

        t1.start();
        t2.start();
    }
}
```

Bài 17: Xử lý dữ liệu với File và Folder

1. **Thư viện cần Import:**
   - import java.io.File;
   - import java.io.FileInputStream;
   - import java.io.FileOutputStream;
   - import java.io.IOException;
   - import java.io.FileNotFoundException;

2. **Khởi tạo file mới:**

```
File file = new File("đường\\dẫn\\namefile.txt");     // Đối tượng file
if (!file.exists()) {
      file.createNewFile();
}
```

3. **Các phương thức của File:**
   - exists(): kiểm tra xem file có tồn tại hay không
   - getName(): lấy tên file (input-file.txt)
   - getParent(): lấy đường dẫn thư mục của file
   - getPath(): đường dẫn đầy đủ
   - isDirectory(): kiểm tra xem là thư mục hay không
   - isFile(): kiểm tra xem là file hay không
   - length(): cỡ file (byte)
   - createNewFile(): tạo ra file mới
   - delete(): xóa file
   - list(): lấy tên file, thư mục chứa trong đường dẫn
   - mkdir(): tạo thư mục
   - renameTo(File dest): đổi tên file

4. **Đọc dữ từ file:**

```
// Đọc dữ liệu từ file
C1: FileInputStream input = new FileInputStream(stringPath);
C2: FileInputStream input = new FileInputStream(File fileObject);
int c = fileInputStream.read();

// Duyệt từng ký tự trong file
while (c != -1) {
     System.out.print((char) c);
```

```
    c = fileInputStream.read();
}

// Đóng file
fileInputStream.close();
```

```
try {
    FileInputStream fileInputStream = new FileInputStream(fileName);
    BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(fileInputStream));
    String line = bufferedReader.readLine();

    while (line != null) {
        line = bufferedReader.readLine();
    }
} catch (FileNotFoundException e) {
    throw new RuntimeException(e);
} catch (IOException e) {
    throw new RuntimeException(e);
}
```

⇨ FileInputStream có thể thay bằng FildeReader

- Một số phương thức khác:
  - read() : đọc một byte từ file
  - read(byte[] array): đọc các byte từ file và lưu kết quả vào mảng
  - read(byte[] array, int start, int length): đọc các byte từ file bắt đầu từ vị trí start cho đến start + length
  - available(): trả về số byte có trong file chưa đọc
  - skip(int): bỏ qua số byte không cần đọc

5. Ghi đè dữ liệu lên file cũ:

- Cách ghi file:
  - C1: true --> cho ghi tiếp
  - FileOutputStream output = new FileOutputStream(String path, boolean value);
  - C2: false --> ghi đè hết lên file cũ
  - FileOutputStream output = new FileOutputStream(String path);
  - C3: Sử dụng đối tượng File:
  - FileOutputStream output = new FileOutputStream(File fileObject);

FileOutputStream

```
// Ghi dữ liệu vào file
FileOutputStream fileOutputStream = new FileOutputStream(file);
String s = "text";
```

FileWriter

```
// Ghi dữ liệu vào file
String s = "text";
FileWriter fileWriter = new FileWriter(file);
```

```
// Ghi đè vào dữ liệu cũ
fileOutputStream.write(s.getBytes());
fileOutputStream.close();
```

```
fileWriter.write(s, true);
// true nếu ghi liền, false nếu xóa đi ghi lại
fileWriter.close();
```

⇨ Xóa sạch dữ liệu cũ từ file và thay bằng dữ liệu mới

- Một số phương thức khác:
  - write(): ghi một byte đến file output stream
  - write(byte[] array): ghi những byte từ một mảng đến the output stream
  - write(byte[] array, int start, int length): ghi length byte đến output stream từ một mảng bắt đầu tại vị trí start của mảng
  - flush(): đẩy dữ liệu trong bộ đệm đến nơi cần lưu trữ

6. File JAR:

- Jar file là file nén bao gồm các file.class được biên dịch và một số file khác nếu có
- Dễ dàng chia sẻ file jar như thư viện cho project khác

7. Xử lý với thư mục Folder:

- Tạo đối tượng thư mục mới:
  - File dir = new File("dirFolder");

- Kiểm tra thư mục tồn tại:

```
// Tạo thư mục mới nếu chưa tồn tại
if (!dir.exists()) {
    dir.mkdir();
}
```

- Gửi tạo file trong thư mục Folder:
  - File file = new File("dirFolder/nameFile.txt");

Bài 18: Tiện ích thú vị Terminal

1. **Kiểm tra Version Java trên CMD:**
    - java -version

2. **Chạy Java bằng Notepad trên CMD:**
    - Tạo file Notepad với câu lệnh Java
    - Lưu tên file bằng NameClass.java
    - Mở terminal di chuyển đến thư mục chứa NameClass.java
    - Gõ lệnh “javac NameClass.java”
    - Thực thi chương trình bằng “java NameClass”

Bài 19: Cấu trúc dữ liệu

1. **Tổng quan về cấu trúc dữ liệu:**