### **What are Methods?**

#### Hình trên bao gồm các nhiệm vụ liên quan đến nông nghiệp.

#### Các công việc như gieo hạt, tưới nước, bón phân kết hợp với nhau để đạt được một mục tiêu duy nhất là cây trồng phát triển tốt.

#### Và khi tất cả các quá trình này làm việc cùng nhau, chúng ta có được đầu ra dưới dạng cây trồng xanh, được nuôi dưỡng.

#### Tương tự, Phương pháp trong lập trình là nhóm các câu lệnh được viết để thực hiện một hoạt động cụ thể.

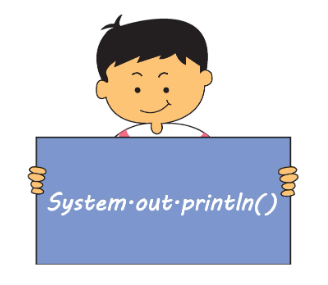
#### Bạn có thể coi các phương thức là hàm nếu bạn đã biết chúng là gì.



#### Nhóm câu lệnh này có thể được sử dụng nhiều lần sau khi được viết.

#### Yup, viết một lần sử dụng bao nhiêu lần tùy thích.

#### Java cung cấp nhiều phương thức tích hợp. Các phương thức tích hợp là những phương thức có trong thư viện Java và có thể được sử dụng trực tiếp.



#### Hãy nhớ phương thức System.out.println() mà chúng ta đã sử dụng?

#### Nó là một phương pháp tích hợp.

#### Nó được định nghĩa ở đâu đó mà một loạt các câu lệnh phải làm việc cùng nhau để thực hiện thao tác in các mục lên màn hình.



#### Bạn có thể tưởng tượng viết cùng một loạt các tuyên bố mỗi khi bạn muốn in một cái gì đó lên màn hình không?

#### Các phương pháp cho phép chúng ta thực hiện các thao tác tương tự mà không cần viết cùng một câu lệnh nhiều lần.

#### Chúng ta cũng có thể viết các phương pháp của riêng mình. Hãy cùng khám phá cách thực hiện.

### **Writing a method**

#### Các phương thức trong Java được viết như mô tả dưới đây

public static int name(parameters){

//statements

}

#### **Public** **static** là các sửa đổi và được sử dụng để xác định phạm vi của các phương thức. Bây giờ hãy coi chúng là điều hiển nhiên.

#### **int** là kiểu trả về. Nó chỉ định loại giá trị được trả về bởi phương thức. Nó có thể là bất kỳ kiểu dữ liệu nào được hỗ trợ. Trong trường hợp này, int có nghĩa là kết quả của việc thực thi các hàm là một số nguyên.

#### **Name** là tên của phương thức.

#### **Parameters** là dữ liệu mà chúng ta có thể truyền cho phương thức để thực hiện các hoạt động.

#### Ví dụ: nếu phương thức thực hiện cộng hai số thì chúng ta cần truyền hai số làm tham số.

#### Và **statements** được đặt trong dấu ngoặc nhọn.

#### Hãy bắt đầu bằng cách viết một phương pháp đơn giản.

public static void hello(){

System.out.println("Congratulations, you have written your first method.");

}

#### Vậy là xong, chúng tôi đã viết phương thức đầu tiên của chúng tôi trong Java.

#### Nhận thấy kiểu trả về **void** mà chúng tôi đã sử dụng. Nó chỉ định rằng phương thức trả về không có gì.

#### Phương pháp này không có tham số và in "Xin chúc mừng, bạn đã viết phương pháp đầu tiên của mình." mỗi khi nó được gọi.

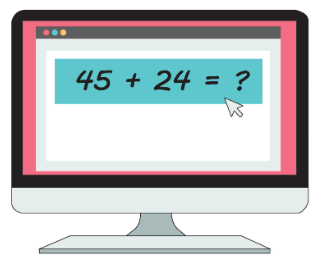


#### Vì vậy, bây giờ nếu chúng ta muốn in một cái gì đó nhiều lần, chúng ta không cần phải viết đi viết lại. System.out.println()

#### Chúng ta chỉ có thể viết một phương thức.

#### Chúng ta sẽ sớm tìm hiểu cách gọi một phương thức.

#### Hãy viết một phương thức nhận các tham số và trả về một cái gì đó.



#### Chúng ta phải viết một phương thức nhận hai tham số, thực hiện thao tác cộng và trả về kết quả.

#### Hãy chia nhỏ điều này và bắt đầu viết phương thức.



#### Giả sử chúng ta đang thực hiện phép **cộng hai int**egers, do đó **return** type sẽ là int.

#### Chúng ta cần truyền hai số nguyên làm tham số mà thao tác cộng có thể được thực hiện. Vì vậy, danh sách tham số của chúng ta sẽ trông giống như trong đó num1 và num2 là hai số nguyên tương ứng.(int num1, int num2)

#### Bây giờ, những gì còn lại?

#### Câu lệnh để cộng các số này và trả về tổng của chúng.

#### Để trả về dữ liệu từ phương thức, chúng ta sử dụng câu lệnh return.

#### Vì vậy, câu lệnh cuối cùng của chúng tôi để thêm hai số và trả về kết quả sẽ như thế nào

return num1 + num2;

#### Hãy kết hợp tất cả những điều này để có được phương pháp của chúng tôi tại chỗ.

public static int sum(int num1, int num2){

return num1 + num2;

}

#### Tuyệt vời, chúng tôi đã viết một phương pháp lấy hai số nguyên và trả về tổng của chúng như là kết quả.

#### Vì vậy, bây giờ nếu chúng ta muốn tính tổng của hai số nguyên bất kỳ, chúng ta sẽ chỉ gọi phương thức này.

#### Chúng ta sẽ học cách gọi một phương thức trong thời gian ngắn.

### **Calling a method**

#### Đủ **các phương pháp** xác định. Hãy xem làm thế nào chúng ta có thể sử dụng chúng.

#### Để sử dụng các phương thức mà chúng ta đã định nghĩa, chúng ta cần thực hiện lệnh gọi đến các phương thức này.



#### Gọi một phương thức sử dụng cú pháp sau:

methodName(parameters);

#### Đó là nó. Để gọi bất kỳ phương thức nào, chúng ta chỉ cần chỉ định tên của nó và các tham số của nó trong ngoặc, nếu có.



#### Bây giờ, bạn có thể đoán những gì chúng tôi đã làm kể từ khi bắt đầu thời gian khi chúng tôi đang viết .. ("?");SystemoutprintlnIsn’t this awesome

#### Chúng tôi thực sự đang thực hiện một cuộc gọi đến **phương thức** **println** và chuyển một tham số mà chúng tôi muốn in ra màn hình.

#### Hãy thực hiện một cuộc gọi đến các phương thức mà chúng tôi đã xác định.

class Main {

public static void hello(){

System.out.println("Congratulations, you have written your first method.");

}

public static void main(String[] args) {

hello();

//call to the hello method

}

}

#### Trên đây sẽ in **Xin chúc mừng**, **bạn** **đã** **viết** **phương pháp** **đầu tiên** **của mình**. trên màn hình đầu ra.

#### Chúng ta có thể gọi phương thức bao nhiêu lần tùy thích và nó sẽ in văn bản cùng một số lần.

class Main {

public static int sum(int num1, int num2){

return num1 + num2;

//returning sum of num1 and num2

}

public static void main(String[] args) {

int add = sum(2,4);

//call to the sum method

int add2 = sum(3,6);

//call to the sum method

System.out.println(add);

System.out.println(add2);

}

}

#### Chúng ta đang gọi phương pháp **sum** ở đây và truyền hai số để thực hiện phép cộng. Chúng tôi đã gọi phương thức hai lần với các tham số khác nhau.

#### Đầu ra của đoạn trích trên sẽ là

6 9

### **Tóm tắt**

#### Phương thức là các khối mã có thể tái sử dụng được viết để thực hiện các tác vụ cụ thể.

#### Các phương thức trong Java được viết theo cách sau

public static int name(parameters){

//statements

}

#### Để gọi bất kỳ phương thức nào, chúng ta cần chỉ định tên và tham số của nó trong ngoặc, nếu có.

### **Công cụ sửa đổi kiểm soát truy cập là gì? Access control modifiers**

#### Mẹ tôi đã hướng dẫn tôi nghiêm ngặt không được bước ra khỏi phòng trong giờ học.

#### Cô ấy đã hạn chế quyền truy cập của tôi vào phần còn lại của ngôi nhà.

#### Và muốn tôi tập trung hoàn toàn vào việc học.

#### Tôi tự hỏi liệu cô ấy có biết về các công cụ sửa đổi kiểm soát truy cập không.

### **Công cụ sửa đổi kiểm soát truy cập**

#### Công cụ sửa đổi kiểm soát truy cập là các từ khóa được sử dụng để hạn chế phạm vi của lớp, thành viên dữ liệu, phương thức hoặc hàm tạo.

#### **Scope** xác định từ nơi các thực thể này có thể được truy cập và nơi chúng không thể.

#### Chúng chỉ định lớp nào có thể truy cập vào một lớp nhất định và các trường, hàm tạo và phương thức của nó.

#### \* Chúng ta sẽ tìm hiểu về các nhà xây dựng sau trong khóa học này.

#### Có 4 công cụ sửa đổi kiểm soát truy cập được cung cấp bởi Java:

1. Default
2. Public
3. Private
4. Protected

### **Default**

#### Khi không có mã xác định truy cập nào được khai báo, nó được cho là có công cụ sửa đổi kiểm soát truy cập **mặc định**.

#### Điều này đúng với lớp, phương thức và thành viên dữ liệu.

#### Các thực thể có mã xác định truy cập mặc định có thể truy cập được trong cùng một gói.

#### Gói là một nhóm các lớp và phương thức bên trong cùng một thư mục. Java sử dụng các thư mục để định nghĩa các gói.

#### Để có mã xác định truy cập mặc định, chúng tôi không cần chỉ định bất cứ điều gì rõ ràng.

class SomeClass{

//body

}

#### Ở đây, lớp **SomeClass** không được khai báo bằng bất kỳ mã xác định truy cập nào, do đó công cụ sửa đổi kiểm soát truy cập của nó là **mặc định**.

#### Lớp này chỉ có thể được truy cập trong cùng một gói, có nghĩa là một lớp khác trong cùng một thư mục có thể truy cập nó, trong khi một lớp trong một thư mục khác không thể.

### **Public**

#### Công cụ sửa đổi kiểm soát truy cập **public** có thể truy cập ở mọi nơi.

#### Nó có phạm vi rộng nhất trong số bốn.

#### Để xác định một thực thể là **public**, chúng tôi sử dụng từ khóa public.

#### Khi được định nghĩa là công khai, thực thể cũng có thể truy cập được bên ngoài gói.

public class ThisClass{

public int thisNum = 10;

}

#### Ở đây, lớp ThisClass được khai báo bằng cách sử dụng từ khóa **public**, do đó công cụ sửa đổi kiểm soát truy cập của nó là công khai.

#### Tương tự, thành viên dữ liệu **thisNum** cũng được khai báo là công khai.

#### Lớp **ThisClass** và thành viên dữ liệu **thisNum** có thể được truy cập ở mọi nơi.

#### Nếu thisNum không công khai, một lớp có thể truy cập nội dung của **ThisClass**, nhưng không phải **thisNum**!

### **Private**

#### Công cụ sửa đổi kiểm soát truy cập **private** chỉ có thể truy cập được trong cùng một lớp.

#### Các thành viên dữ liệu và phương thức được khai báo là **private** có thể truy cập được trong cùng một lớp.

#### Nó có phạm vi ít nhất trong số bốn.

#### Để xác định một thực thể là **private**, chúng tôi sử dụng từ khóa private.

class ThisClass{

private int thisNum = 10;

}

#### Ở đây, thành viên dữ liệu **thisNum** được khai báo bằng từ khóa riêng tư, do đó công cụ sửa đổi kiểm soát truy cập của nó là **private**.

#### Do đó, nó chỉ có thể được truy cập trong lớp **ThisClass**.

### **Protected**

#### Công cụ sửa đổi kiểm soát truy cập được **protected** có thể truy cập được trong cùng một gói cũng như bên ngoài nhưng chỉ sử dụng kế thừa.

#### Chỉ các phương thức, thành viên dữ liệu và hàm xây dựng mới có thể được khai báo là **protected**.

#### Các lớp không thể được tuyên bố là **protected**.

#### \* Chúng tôi sẽ tìm hiểu thêm về thừa kế trong khóa học.

#### Để xác định một thực thể là được bảo vệ, chúng tôi sử dụng từ khóa **được protected**.

class ThisClass{

protected int thisNum = 10;

}

#### Ở đây, thành viên dữ liệu **thisNum** được khai báo bằng từ khóa được bảo vệ, do đó công cụ sửa đổi kiểm soát truy cập của nó **protected**.

#### Do đó, nó có thể được truy cập trong cùng một gói và bên ngoài nhưng chỉ khi một số lớp kế thừa lớp **ThisClass**.

### **Tóm tắt**

#### Công cụ sửa đổi kiểm soát truy cập là các từ khóa chỉ định lớp nào có thể truy cập vào một lớp nhất định và các trường, hàm tạo và phương thức của nó.

#### Khi không có mã xác định truy cập nào được khai báo, nó được cho là có công cụ sửa đổi kiểm soát truy cập **mặc định**. Các thực thể có mã xác định truy cập mặc định có thể truy cập được trong cùng một gói.

#### Công cụ sửa đổi kiểm soát truy cập **công cộng** có thể truy cập ở mọi nơi.

#### Công cụ sửa đổi kiểm soát truy cập **riêng tư** chỉ có thể truy cập được trong cùng một lớp.

#### Công cụ sửa đổi kiểm soát truy cập được **bảo vệ** có thể truy cập được trong cùng một gói cũng như bên ngoài nhưng chỉ sử dụng kế thừa.

### **What is a constructor?**

#### Constructor là một phương thức có cùng tên với lớp.

public class learningConstructor{

public learningConstructor(){

//Constructor

//statements

}

}

#### Trong đoạn mã trên là một hàm tạo có cùng tên với lớp.learningConstructor()



#### Các hàm xây dựng thường được gọi là các loại phương thức đặc biệt trong Java.

#### Mỗi lớp đều có một hàm tạo. Một hàm tạo "xây dựng" lớp riêng của nó.

#### Các nhà xây dựng không được coi là thành viên của lớp.



#### Nếu chúng ta không viết một hàm tạo, trình biên dịch java sẽ viết một cái cho chúng ta.

#### Hàm tạo mà trình biên dịch java viết cho chúng ta được gọi là hàm tạo mặc định và nó được chèn vào mã trong quá trình biên dịch.

#### Nhưng, nếu chúng ta viết một hàm tạo, hàm tạo mặc định không bao giờ được tạo.

#### Mỗi khi một thể hiện của đối tượng được tạo, **hàm tạo** được gọi.

#### Một hàm tạo được sử dụng để cung cấp các giá trị ban đầu cho các trường lớp khi một đối tượng được tạo.

#### Cũng giống như bất kỳ phương thức nào khác, hàm tạo có thể có hoặc không có bất kỳ tham số nào.

#### Nếu một hàm tạo có các tham số, nó được gọi là hàm tạo **tham số**.

### **Constructors in action**

#### Hãy bắt đầu bằng cách viết một hàm tạo đơn giản.

public class learningConstructor{

public learningConstructor(){

System.out.println("Hi! I am a constructor :D");

}

}

#### Chúng tôi đã viết một hàm tạo đơn giản in "Xin chào! Tôi là một nhà xây dựng :D" Mỗi khi một thể hiện mới của đối tượng được tạo.

public class LearningConstructor{

public LearningConstructor(){

System.out.println("Hi! I am a constructor :D");

}

public static void main(String[] args) {

LearningConstructor learning1 = new LearningConstructor();

}

}

#### Chúng ta đã tạo ra một instance mới của object. Chương trình trên về thực hiện cho đầu ra.

Chào bạn! Tôi là một nhà xây dựng :D



#### Chúng ta đã thấy cách một hàm tạo không có tham số hoạt động.

#### Hãy khám phá cách chúng ta có thể khởi tạo các thành viên lớp bằng cách sử dụng hàm tạo.

#### Đối với điều này, chúng tôi sẽ sử dụng một **hàm tạo tham số**.

#### Bạn muốn biết làm thế nào? vòi! vòi!

public class LearningConstructor{

String name;

String type;

public LearningConstructor(String newName, String newType){

this.name = newName;

this.type = newType;

}

}

#### Trong đoạn mã trên, chúng ta có một hàm tạo có tên LearningConstructor() có hai tham số **newName** và **newType** đều thuộc loại String.



#### Thấy có gì lạ không? this.name? this.type?

#### **đây** là một từ khóa trong Java.

#### Nó được sử dụng để tham chiếu trường hợp hiện tại của phương thức mà nó được sử dụng.

#### Vậy

helloFunction(int a){

this.someVariable = a;

}

#### Ở đây, đề cập đến thể hiện của phương thức mỗi khi nó được gọi.this.someVariable

#### Trở lại, trở lại. Chúng tôi đã tìm hiểu về các nhà xây dựng tham số.

public class LearningConstructor{

String name;

String type;

public LearningConstructor(String newName, String newType){

this.name = newName;

this.type = newType;

}

}

#### Hãy tạo một đối tượng kiểu LearningConstructor.

public class LearningConstructor{

String name;

String type;

public LearningConstructor(String newName, String newType){

this.name = newName;

this.type = newType;

}

public static void main(String[] args){

LearningConstructor learningConstructor1 = new LearningConstructor(“Hi”, “Default”);

}

}

#### Tuyệt vời, chúng ta có một đối tượng thuộc loại LearningConstructor. Hãy viết một phương thức hiển thị các giá trị được gán cho các thành viên của lớp.

public class LearningConstructor{

String name;

String type;

public LearningConstructor(String newName,String newType){

this.name = newName;

this.type = newType;

}

void display(){

System.out.println("Name: "+name+" Type: "+type);

}

public static void main(String[] args){

LearningConstructor obj1 = new LearningConstructor("Hi", "Default"); obj1.display();

}

}

#### Đầu ra của đoạn trích trên sẽ là

Tên: Xin chào Kiểu: Mặc định

#### Phương thức chính tạo ra một thể hiện của lớp LearningConstructor, đặt tên thành "Hi" và Type thành "Default". Sau đó chúng ta gọi phương thức display() của đối tượng obj1.

#### Wow! Chúng tôi đã học được rất nhiều về các nhà xây dựng.

#### Bắt đầu từ hàm tạo, hàm tạo mặc định, hàm tạo không có tham số cho đến hàm tạo tham số là gì.

#### Hãy đưa kiến thức này vào thử nghiệm.

### **Tóm tắt**

#### Hàm tạo là một phương thức có cùng tên với lớp.

#### Nếu chúng ta không viết một hàm tạo, trình biên dịch java sẽ viết một hàm cho chúng ta, được gọi là hàm tạo mặc định.

#### Mỗi khi một thể hiện của đối tượng được tạo, hàm tạo được gọi.

#### Một hàm tạo được sử dụng để cung cấp các giá trị ban đầu cho các trường lớp khi một đối tượng được tạo.

#### Nếu một hàm tạo có các tham số, nó được gọi là hàm tạo tham số.

### **What is inheritance?**

#### Tại sao tôi trông rất giống cha tôi? Mũi của tôi cũng giống hệt anh ấy.

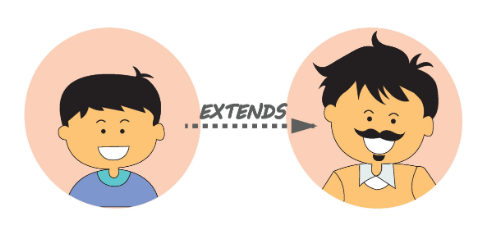
#### Đó là bởi vì bạn đã thừa hưởng các tính năng của anh ấy.

#### **Kế thừa** là khi một lớp lấy được một số thuộc tính từ một lớp khác.

#### Cũng giống như con người thừa hưởng những đặc điểm từ cha mẹ.

#### **Lớp** kế thừa các thuộc tính được gọi là lớp **con**.

#### Và lớp mà từ đó các thuộc tính được kế thừa được gọi là **lớp** **cha**.



#### Để kế thừa các thuộc tính của một lớp, **từ khóa mở rộng** được sử dụng.

class ParentClass{

//body

}

class ChildClass extends ParentClass{

//body

}

#### Đây là cách chúng ta **mở rộng** một lớp. Ở đây **ChildClass** là một lớp kế thừa từ **ParentClass**.

#### Một lớp **con** kế thừa tất cả các thành viên của **lớp** **cha**.

#### Một đối tượng được tạo ra từ **lớp** **con** kiểu có thể được sử dụng để truy cập tất cả các thành viên của lớp **cha**.

#### **Hàm tạo** không bao giờ được kế thừa.

### **Inheritance in action**

#### Hãy cùng tìm hiểu thêm về thừa kế với sự trợ giúp của ví dụ sau.

public class AnotherClass{

public void add(int num1, int num2){

System.out.println(num1+num2);

}

}class Main extends AnotherClass{

public static void main(String[] args) {

Main obj1 = new Main(); //obj1 of type Main

obj1.add(3,5); //Calling add method inherited from AnotherClass

}

}

#### Trong đoạn mã trên, chúng ta có một class **Main** kế thừa class **AnotherClass**.

#### Bạn có đoán được đầu ra không?

#### Chúng ta đang gọi phương thức add được khai báo trong **AnotherClass** từ lớp **Main**.

#### Và chúng tôi đang truyền **3** và **5** làm tham số.

#### Vì vậy, nếu câu trả lời của bạn là **8** thì bạn thật tuyệt vời.

### **Types of inheritance**

#### Các lớp có thể kế thừa từ các lớp khác và theo những cách khác nhau.

#### Điều này đưa chúng ta đến nghiên cứu về các loại thừa kế khác nhau.

#### Có ba loại kế thừa trong Java - Kế thừa đơn, Kế thừa đa cấp và Kế thừa phân cấp.

### **Thừa kế đơn lẻ**

#### Đây là loại thừa kế đơn giản nhất.

#### Trong đó, một lớp con kế thừa các thuộc tính của lớp cha.

class Parent{

//body

}

class child extends parent{

//body

}

#### Chỉ vậy thôi.

#### Bạn có thể đoán loại thừa kế mà chúng tôi đã sử dụng trong ví dụ bổ sung mà chúng tôi đã thấy gần đây không?

#### Nếu câu trả lời của bạn là **thừa kế** **duy nhất** thì hãy tiếp tục lăn.

#### Trong Kế thừa đơn, một và chỉ một lớp có thể mở rộng một lớp khác.

#### Một lớp phụ huynh duy nhất và một lớp con duy nhất.

### **Kế thừa đa cấp**

#### Trong kế thừa đa cấp có một chuỗi các lớp.

#### Từ hình trên chúng ta có thể thấy ClassB là một lớp con của **ClassC** và **ClassA** là một lớp con của **ClassB**.

#### Nó có thể được biểu diễn cú pháp dưới dạng

class A{

//body

}

class B extends A{

//body

}

class C extends B{

//body

}

#### Ở đây, lớp A là lớp mẹ của lớp B và lớp B là lớp mẹ của lớp C.

#### Không có giới hạn của chuỗi này. Có thể có một lớp D mở rộng lớp C và vân vân.

### **Kế thừa phân cấp**

#### Trong **kế thừa** **phân cấp**, nhiều hơn một lớp mở rộng các thuộc tính của một lớp.

#### Đó là, một lớp cha mẹ duy nhất có nhiều lớp con.

#### Có thể có nhiều lớp học trẻ em như chúng ta muốn.

#### Nhưng tất cả các lớp này chỉ được có một lớp cha mẹ duy nhất.

class A{

//body

}

class B extends A{

//body

}

class C extends A{

//body

}

#### Ở đây, lớp A là lớp mẹ của lớp B và lớp C.

#### Có thể có một lớp D cũng mở rộng lớp A và vân vân.

#### Trong Java, một lớp duy nhất không thể kế thừa từ nhiều hơn một lớp.

#### Loại thừa kế này được gọi là **Đa** **thừa kế**.

#### Và **Multiple** **inheritance** không được hỗ trợ trong Java.

### **Tóm tắt**

#### Kế thừa là khi một lớp lấy được một số thuộc tính từ một lớp khác.

#### **Lớp** kế thừa các thuộc tính được gọi là lớp **con** và lớp mà từ đó các thuộc tính được kế thừa được gọi là lớp **cha**.

#### Có ba loại kế thừa trong Java - Kế thừa đơn, Kế thừa đa cấp và Kế thừa phân cấp.

#### **Đa** **kế thừa** không được Java hỗ trợ.

### **What is Overloading?**

#### Quá tải là gì?

#### Nhưng để giải quyết các cá nhân, chúng tôi sử dụng tên cùng với họ để xác định duy nhất cá nhân đó.

#### Quá tải khá giống với những gì chúng ta đã thấy ở trên.

#### **Quá tải** là khi hai hoặc nhiều phương thức trong một lớp có cùng tên.

#### Nhưng có các thông số khác nhau.

#### Các tham số có thể khác nhau về số lượng tham số hoặc loại.



#### Nhưng, tôi có thể viết nhiều phương thức thực hiện cùng một tác vụ với các tham số khác nhau.

#### Nghiêm túc? Tại sao?

#### Quá tải ngăn cản chúng ta viết nhiều phương thức thực hiện cùng một tác vụ.

#### Chúng ta có thể làm quá tải các phương thức và thao tác với loại tham số mà chúng chấp nhận

#### Hãy xem điều này trong hành động.

class Main {

public static void dontOverloadMe(){

System.out.println("Please, don't.");

}

public static void dontOverloadMe(String laugh){

System.out.println("Oops! Couldn't hold it for long. "+ laugh);

}

}

#### Trong lớp Main, chúng ta có hai phương thức có cùng tên, một trong số đó không chấp nhận tham số và một phương thức khác chấp nhận một tham số.

#### Vì vậy, chúng ta có thể nói rằng phương pháp thứ hai làm quá tải phương pháp thứ nhất.

#### Nhưng làm thế nào để xác định cái nào được gọi là khi nào?

#### Trình biên dịch thực hiện điều đó cho bạn.

#### Khi bạn gọi một phương thức, nó sẽ kiểm tra các tham số được truyền nếu có và theo đó thực hiện phương thức tương ứng.

#### Hãy thử gọi phương thức quá tải.

class Main {

public static void dontOverloadMe(){

System.out.println("Please, don't.");

}

public static void dontOverloadMe(String laugh){

System.out.println("Oops! Couldn't hold it for long. "+ laugh);

}

public static void main(String[] args) {

dontOverloadMe();

//calling the function without any parameters

}

}

#### Đầu ra của đoạn trích trên sẽ là

Làm ơn, đừng.

#### Tuyệt vời! Nhưng nếu tôi vượt qua một tham số với nó thì sao?

class Main {

public static void dontOverloadMe(){

System.out.println("Please, don't.");

}

public static void dontOverloadMe(String laugh){

System.out.println("Oops! Couldn't hold it for long. "+ laugh);

}

public static void main(String[] args) {

dontOverloadMe("HaHaHa"); //calling the function with a parameter

}

}

#### Đầu ra của đoạn trích trên sẽ là

Rất tiếc! Không thể nhịn được lâu. HaHaHa

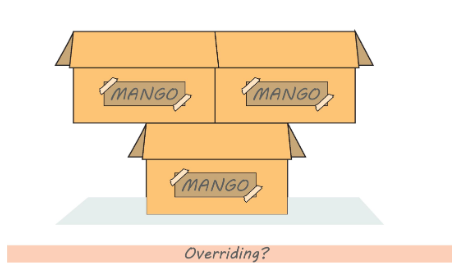
#### Chết tiệt, vâng! Đó là cách nó được thực hiện.

### **What is Overriding?**

#### Một số thành viên trong một gia đình có biệt danh vì tên thật rất dài hoặc khó gọi.

#### Do đó, trong trường hợp này, biệt danh đang được sử dụng để xác định cùng một cá nhân.

#### Ghi đè hoàn toàn phù hợp với ví dụ trên.



#### Ghi đè là khi hai hoặc nhiều phương thức trong các lớp riêng biệt có mối quan hệ con-cha mẹ có cùng tên.

#### Và cũng có thông số tương tự.

#### Nhưng, hai phương pháp không được viết trong cùng một lớp.

#### Phương thức ghi đè lên phương thức trước được viết trong một lớp con.

#### Điều này cho phép chúng tôi cung cấp các thông số kỹ thuật trong khi thực hiện cùng một chức năng trong một lớp con.

#### Chúng tôi đã tìm hiểu về các lớp học trẻ em và kế thừa.

#### Điều quan trọng cần lưu ý là ghi đè diễn ra khi một phương thức được khai báo trong lớp cha được viết lại trong một lớp con có cùng tham số.

#### Hãy xem ghi đè được thực hiện như thế nào.

class ParentClass{

public void oldMethod(){

System.out.println("Parents rock.");

}

}class ChildClass extends ParentClass{

public void oldMethod(){

System.out.println("Children rock. ");

}

}

#### Ở đây, **ChildClass** là một lớp được kế thừa từ ParentClass có cùng hàm oldFunction() được khai báo trong **ParentClass**.

#### Bạn nghĩ điều gì sẽ xảy ra nếu chúng ta gọi phương thức **oldMethod**?

class ParentClass{

public void oldMethod(){

System.out.println("Parents rock.");

}

}class ChildClass extends ParentClass{

public void oldMethod(){

System.out.println("Children rock.");

}

}

public class Main{

public static void main(String [] args){

ParentClass obj1 = new ChildClass();

obj1.oldMethod();

//calling method oldMethod

}

}

#### Bạn có đoán được đầu ra của đoạn trích trên không?

Trẻ em đá.

### **Tóm tắt**

#### **Quá tải** là khi hai hoặc nhiều phương thức trong một lớp có cùng tên nhưng các tham số khác nhau.

#### Quá tải ngăn cản chúng ta viết nhiều phương thức thực hiện cùng một nhiệm vụ.

#### **Ghi đè** là khi hai hoặc nhiều phương thức trong một lớp có cùng tên và cùng tham số.

#### Phương thức ghi đè lên phương thức trước được viết trong một lớp con.

### **What is Polymorphism?**

#### Nghĩa đen của đa hình là có các hình thức hoặc **hình** dạng khác nhau.

#### Một ví dụ thực tế về tính đa hình có thể là mặt trăng khi nhìn từ trái đất.

#### Tính chất của mặt trăng có hình dạng khác nhau, tức là các giai đoạn của mặt trăng.

#### Trong Lập trình hướng đối tượng, đa hình là một cách để thực hiện một hành động duy nhất theo những cách khác nhau.



#### Trong Java, tính đa hình có thể được xem là khả năng của các phương thức để làm những việc khác nhau dựa trên đối tượng mà nó đang hoạt động.

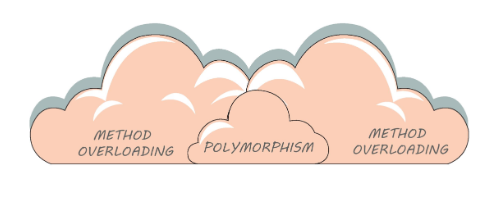
#### Điều đó nghe có vẻ giống với bất cứ điều gì?

#### Bạn có nhớ quá tải phương pháp không?



#### **Phương pháp** **quá tải** rơi vào đa hình.

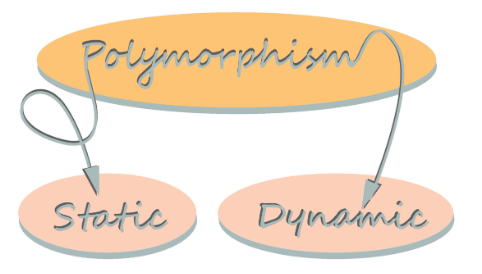
#### Vì nó cho phép chúng ta định nghĩa một thứ và sử dụng nó theo các cách khác nhau.



#### Nhưng chờ đã, nếu **quá tải** phương thức là một loại đa hình thì còn phương thức **ghi đè thì sao**?

#### Phương thức ghi đè cũng cho phép chúng ta sử dụng cùng một phương thức theo những cách khác nhau.

#### Do đó, phương pháp ghi đè cũng rơi vào đa hình.



#### Đa hình có hai loại -

* Đa hình tĩnh
* Đa hình động.

### **Đa hình tĩnh**

#### **Đa hình tĩnh** còn được gọi là đa **hình** thời gian biên dịch đạt được thông qua quá tải phương pháp.

#### Tại thời điểm biên dịch, Java biết phương thức nào cần gọi bằng cách kiểm tra chữ ký phương thức do đó nó được gọi là đa hình thời gian biên dịch.

#### Bạn có nhớ ví dụ quá tải mà chúng tôi đã nghiên cứu không?

class Main {

public static void dontOverloadMe(){

System.out.println("Please, don't.");

}

public static void dontOverloadMe(String laugh){

System.out.println("Oops! Couldn't hold it for long. "+ laugh);

}

}

#### Mọi hoạt động quá tải phương pháp cuối cùng là một đa hình, giống như phương pháp trên.

### **Đa hình động**

#### Trong khi đa hình động hoặc đa hình thời gian chạy đạt được thông qua ghi đè phương pháp.

#### Trong trường hợp ghi đè phương thức gọi được xác định trong thời gian chạy, do đó nó được gọi là thời gian chạy hoặc đa hình động.

class ParentClass{

public void oldMethod(){

System.out.println("Parents rock.");

}

}

class ChildClass extends ParentClass{

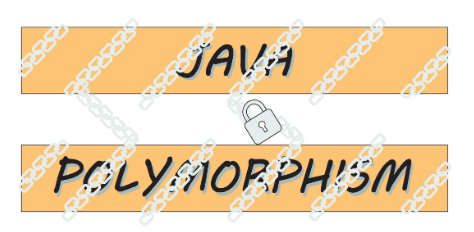
public void oldMethod(){

System.out.println("Children rock. ");

}

}

#### Đoạn mã này trong đó **ChildClass** mở rộng **ParentClass** và bị ghi đè là một ví dụ về đa hình thời gian chạy.oldMethod()



#### Đa hình là không thể tránh khỏi trong Java.

#### Quá tải, ghi đè, mở rộng các lớp cuối cùng đều góp phần vào tính đa hình.

#### Wow! Chúng tôi đã đi một chặng đường dài từ nơi chúng tôi bắt đầu. Nghe nói điều đó trước đây?

#### Hãy đặt những điều chúng ta đã học được để kiểm tra.

### **Tóm tắt**

#### **Đa hình** là một cách để thực hiện một hành động duy nhất theo những cách khác nhau.

#### **Đa** hình có hai loại - Đa hình **tĩnh** và **Đa hình** **động**.

#### Đa hình tĩnh còn được gọi là đa hình thời gian biên dịch đạt được thông qua **quá tải** **phương pháp**.

#### Đa hình động hoặc đa hình thời gian chạy đạt được thông qua ghi **đè** **phương thức**.

### **What is abstraction?**

#### Hãy tưởng tượng bạn đang lái một chiếc xe hơi.

#### Trong khi lái xe bạn tăng tốc, áp dụng nghỉ ngơi, sang số, phải không?

#### Bạn có bao giờ nghĩ về động cơ xe hơi tạo ra năng lượng để di chuyển chiếc xe phía trước không?



#### Không.

#### Bạn chỉ cần tập trung vào những điều quan trọng trong khi lái xe. Biết cách thức hoạt động của động cơ sẽ không giúp bạn.

#### Bạn nhớ những thứ quan trọng và quên đi các chi tiết nền.

#### Đây là những gì **trừu tượng** chính xác là về.



#### Trừu tượng là một quá trình ẩn các chi tiết triển khai khỏi người dùng và chỉ cung cấp chức năng.

#### Nói cách khác, trừu tượng là một quá trình che giấu các chi tiết nền và tập trung vào các chi tiết cần thiết.

#### Bằng cách này, người dùng sẽ biết câu trả lời cho **những gì** nhưng không phải **là làm thế nào** của nó.

#### Điều này cũng có thể được xem là ẩn dữ liệu hoặc chỉ hiển thị các chi tiết cần thiết.

#### Điều này trở nên cần thiết vì không phải mọi người dùng đều phải xem hoặc biết mọi thứ.

#### Ví dụ: một người đang sử dụng phần mềm không cần xem bảng quản trị. Điều đó không có ích gì cho anh ta.

#### Ngoài ra, bảo mật có nguy cơ nếu không có sự trừu tượng.

### **Implementing abstraction**

#### Trong Java, ion trừu tượng có thể đạt được bằng cách sử dụng **các giao diện** và **các lớp** **trừu tượng**.

#### Một lớp trừu tượng là một lớp không hoàn chỉnh, để sử dụng nó chúng ta cần mở rộng nó.

#### Mở rộng lớp trừu tượng làm cho nó hoàn chỉnh.

#### Một lớp trừu tượng được khai báo bằng cách sử dụng từ khóa **trừu tượng**.

#### Một lớp trừu tượng có thể chứa hoặc không chứa các **phương pháp** **trừu tượng**.

#### Một phương pháp **trừu tượng** là một **phương pháp** không có cơ thể.

#### Nó chỉ được khai báo trong một lớp trừu tượng và được định nghĩa trong lớp mở rộng lớp trừu tượng.

#### Một lớp có ít nhất một phương thức trừu tượng cần được khai báo là một lớp trừu tượng.

public abstract class ClassName{

//Members

}

#### Không có sự khác biệt trong khi viết một lớp và một lớp trừu tượng, bổ sung duy nhất là từ khóa **trừu tượng**.

#### Chúng ta có thể mở rộng lớp trừu tượng này giống như cách chúng ta mở rộng các lớp bình thường.

public class AnyClass extends ClassName{

//Members

}

#### Cứ như vậy.



#### Phương pháp trừu tượng là một phương pháp chỉ có khai báo của phương thức.

#### Định nghĩa của phương thức được chứa bởi lớp con mở rộng lớp chứa phương thức trừu tượng.



#### Để khai báo một phương pháp trừu tượng, từ khóa **trừu tượng** được sử dụng.

public abstract returnType methodName();

#### Việc thực hiện phương thức có thể được thực hiện trong bất kỳ lớp nào mở rộng lớp trừu tượng có chứa phương thức này.

#### Chúng ta có thể đạt được sự trừu tượng hoàn toàn bằng cách sử dụng các giao diện.

#### Chúng ta sẽ tìm hiểu thêm về các giao diện trong khóa học.

#### Hãy kết thúc điều này và tập trung vào những điều quan trọng, những câu hỏi sắp tới.

### **Tóm tắt**

#### Trừu tượng là một quá trình che giấu các chi tiết nền và tập trung vào các chi tiết cần thiết.

#### Trong Java, ion trừu tượng có thể đạt được bằng cách sử dụng **các giao diện** và **các lớp** **trừu tượng**.

#### Trong khi viết một lớp trừu tượng hoặc một phương pháp trừu tượng, từ khóa **trừu tượng** được sử dụng.

#### Một phương pháp **trừu tượng** là một **phương pháp** không có cơ thể.

#### Nó chỉ được khai báo trong một lớp trừu tượng và được định nghĩa trong lớp mở rộng lớp trừu tượng.

### **What is encapsulation?**

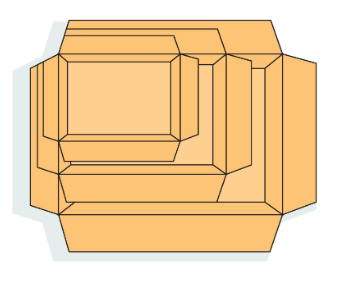
#### Hãy tưởng tượng bạn có một chai nước chứa đầy nước đến vành.

#### Có hai viên bi hiện diện bên trong chai nước.

#### Bạn có thể kéo những viên bi này ra mà không làm đổ nước không?

#### Điều gì sẽ xảy ra nếu bạn được tặng một cặp kẹp?

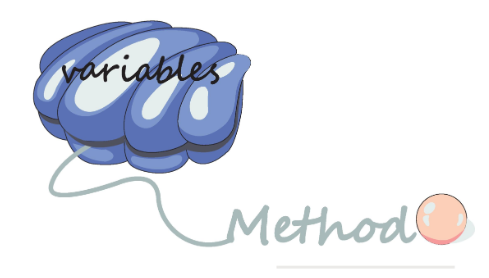
#### Bạn có thể, phải không?



#### Đóng gói là một khái niệm trong Lập trình hướng đối tượng mô tả ý tưởng kết hợp các biến và phương thức lại với nhau.

#### Nó mang lại ý tưởng ẩn dữ liệu vào bảng.

#### Các biến trong lớp chỉ có thể được truy cập thông qua các phương thức được khai báo bên trong lớp.



#### Các biến được khai báo là riêng tư.

#### Và các phương thức để truy cập chúng dưới dạng công khai để chúng có thể được truy cập từ bên ngoài lớp.

#### Sau đó, chỉ các phương thức này có thể được sử dụng để đặt hoặc lấy các giá trị của các biến.

#### Đóng gói cung cấp một không gian được bảo vệ cho các thành viên dữ liệu và hạn chế quyền truy cập bên ngoài lớp.

#### Hãy xem làm thế nào chúng ta có thể đạt được đóng gói trong Java.

public class Student{

private String name; //private data members

private int roll\_no;

public void setName(String newName){

//public methods

name = newName;

}

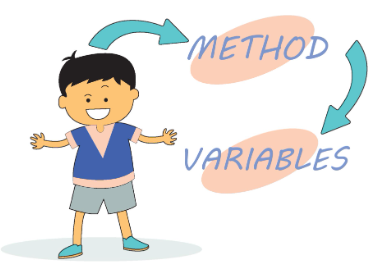
public void setRollNo(int roll){

roll\_no = roll;

}

}

#### Trong ví dụ trên, chúng ta có thể thấy **rằng name** và **roll\_no** được khai báo là private và do đó chúng không thể được truy cập bên ngoài class.



#### Nhưng các phương thức setName() và setRollno() được khai báo là public.

#### Do đó, các phương thức này có thể được truy cập từ bên ngoài lớp.

#### Chúng ta có thể gọi các phương thức này và truyền các giá trị của **newName** và **roll**, sau đó được gán cho các biến **name** và **roll\_no**.

#### Do đó, chúng ta không thể thay đổi giá trị của các biến này mà không truy cập các phương thức đó.

//File: Student.java

public class Student{

private String name; //private data members

private int roll\_no;

public void setName(String newName){

//public methods

name = newName;

}

public void setRollNo(int roll){

roll\_no = roll;

}

}

//File: Main.java

public class Main {

public static void main(String[] args) {

Student obj1 = new Student();

obj1.setName("Rocket");

//Calling method setName

}

}

#### Ở đây, chúng ta đang gọi phương thức **setName** và thiết lập giá trị của **tên** biến.

  
**Tên** và **roll\_no** biến được khai báo là riêng tư và do đó không thể truy cập được.

#### Do đó, chúng ta có các phương thức public **setName** và **setRollNo** để gán giá trị cho các biến này.

#### Ở đây, chúng ta đã gói các biến và phương thức lại với nhau, không gì khác ngoài việc đóng gói.

#### Hãy thử định nghĩa một phương thức khác hiển thị giá trị của tên.

//File: Student.java

public class Student{

private String name;

public void setName(String newName){

name = newName;

}

public void dispName(){

System.out.println(name);

}

}

//File: Main.java

class Main {

public static void main(String[] args) {

Student obj1 = new Student();

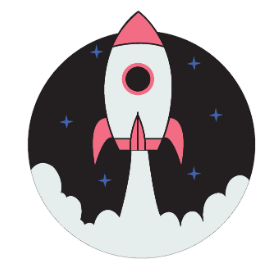
obj1.setName("Rocket");

obj1.dispName();

}

}

#### Bạn có đoán được đầu ra của đoạn trích trên không?



#### Ở đây chúng ta có một phương thức hiển thị giá trị của tên biến.dispName()

#### Sau khi chúng ta gọi phương thức với **Rocket** là tham số, chúng ta đang gọi phương thức.setName()dispName()

#### Do đó, đầu ra của đoạn mã sẽ là

Tên lửa

### **Tóm tắt**

#### Đóng gói là một khái niệm trong OOP mô tả ý tưởng kết hợp các biến và phương thức lại với nhau.

#### Các biến được khai báo là riêng tư.

#### Và các phương thức để truy cập chúng dưới dạng công khai để chúng có thể được truy cập từ bên ngoài lớp.

#### Đóng gói cung cấp một không gian được bảo vệ cho các thành viên dữ liệu và hạn chế quyền truy cập bên ngoài lớp.

### **What is an interface?**

#### Hãy xem xét chúng ta phải viết các lớp học cho ô tô hiển thị các tính năng riêng của chúng.

#### Chúng tôi sẽ yêu cầu các lớp khác nhau cho những chiếc xe khác nhau.

#### Nhưng có rất nhiều tính năng giống nhau trên những chiếc xe này.

#### Chúng tôi giữ tất cả các tính năng tương tự này trong một lớp duy nhất, giống như một mẫu và mở rộng lớp này cho những chiếc xe cụ thể.

#### Giao diện cũng hoạt động theo cách tương tự.



#### Giao diện hoạt động như một tài liệu tham khảo cho các lớp khác.

#### Giao diện là một tập hợp các **phương pháp** **trừu tượng** và **hằng số**.

#### Các phương thức và hằng số này có thể được sử dụng bởi các lớp thực hiện giao diện.



#### Giao diện được sử dụng để đạt được sự trừu tượng hoàn toàn.

#### Ngoài ra, các giao diện có thể được sử dụng để đạt được nhiều kế thừa trong java, điều này là không thể khác.



#### Một giao diện được viết chính xác như một lớp.

#### Hãy xem làm thế nào chúng ta có thể khai báo một giao diện.

### **Writing an Interface**

#### Để khai báo một giao diện, **interface** từ khóa được sử dụng.

public interface InterfaceName{

//Members

}

#### Một giao diện trừu tượng, không cần sử dụng từ khóa **trừu tượng** để khai báo giao diện.



#### Ngoài ra, các phương thức được khai báo bên trong giao diện không cần **từ khóa trừu tượng**. Chúng trừu tượng theo mặc định.

#### Và các phương thức trong một giao diện được công khai theo mặc định.

#### Hãy bắt đầu bằng cách viết một giao diện đơn giản.

public interface Student{

public void dispName();

public void dispRoll();

}

#### Đây là một giao diện **Student** chứa hai phương pháp trừu tượng và .dispName()dispRoll()

### **Using an interface**

#### Để sử dụng giao diện mà chúng ta đã xác định, chúng ta cần **implement** nó.

#### Đúng vậy, các giao diện được **triển khai** và không **được mở rộng**.



#### Chúng tôi sử dụng **implements** khóa để làm như vậy.

#### Một lớp có thể thực hiện nhiều hơn một giao diện cùng một lúc.

#### Hãy xem làm thế nào chúng ta có thể triển khai một giao diện.

public class ClassName implements InterfaceName{

//Member definitions

}

#### Tên của lớp được theo sau bởi **các menu** từ khóa, theo sau là tên của giao diện.

#### Hãy triển khai giao diện sinh viên mà chúng ta đã định nghĩa trước đây.

public class Student1 implements Student {

public void dispName(){

//defining abstract method dispName

System.out.println("Starlord");

}

public void dispRoll(){

//defining abstract method dispRoll

System.out.println("25");

}

public static void main(String[] args) {

Student1 obj1 = new Student1();

obj1.dispName();

obj1.dispRoll();

}

}

#### Ở đây, chúng ta đã định nghĩa các phương thức trừu tượng mà chúng ta khai báo trong giao diện Student.

#### Bạn có đoán được đầu ra của đoạn trích trên không?

#### Đầu ra sẽ là

Lãnh chúa 25

#### Một lớp có thể **implement** một giao diện nhưng giống như một lớp có thể mở rộng các lớp khác, một giao diện cũng có thể **extend** các giao diện khác.

#### Hãy xem làm thế nào. Để làm như vậy, chúng tôi sử dụng **extends** từ khóa rất riêng của chúng tôi.

#### public interface InterfaceName extends SomeOtherInterface{

#### //Members

#### }

#### 

#### Vì vậy, nếu chúng ta muốn mở rộng giao diện Student mà chúng ta đã khai báo, chúng ta sẽ viết:

#### public interface NewStudent extends Student{

#### public void admissionNumber();

#### }

#### 

#### Như vậy, New Student có một phương thức kế thừa hai phương thức khác từ giao diện **Student**. admissionNumber()

### **Tóm tắt**

#### Giao diện là một tập hợp các **phương pháp** **trừu tượng** và **hằng số**.

#### Giao diện được sử dụng để đạt được sự trừu tượng hoàn toàn và đa kế thừa.

#### Một giao diện trừu tượng, không cần phải sử dụng từ khóa **trừu tượng** để khai báo giao diện hoặc các phương thức bên trong nó.

#### Một lớp có thể **thực hiện** một giao diện nhưng giống như một lớp có thể mở rộng các lớp khác, một giao diện cũng có thể **mở rộng** các giao diện khác.

#### Một lớp có thể thực hiện nhiều hơn một giao diện cùng một lúc.

### 

### 

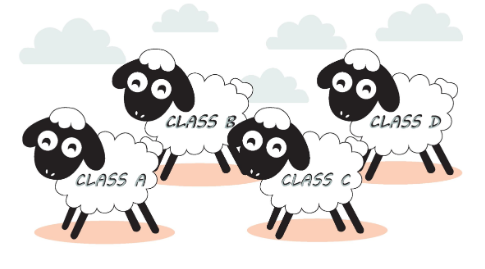
### **What are packages?**

#### Thư viện là gì?

#### Wikipedia định nghĩa thư viện là một tập hợp các nguồn thông tin và các tài nguyên tương tự, được truy cập bởi một cộng đồng xác định để tham khảo hoặc mượn.

#### Thư viện là một nơi chứa hàng tấn sách và tài nguyên mà chúng ta có thể đến và truy cập.

#### Các gói trong Java có phần giống nhau.



#### Một gói trong Java là một nhóm các loại **lớp**, **giao diện** và **gói** **con** tương tự.

#### Các gói chứa bên trong một gói khác được gọi là **gói** **phụ**.

#### Các gói cho phép chúng tôi duy trì mã của mình và cải thiện hiệu quả bằng cách thúc đẩy **khả năng tái sử dụng**.



#### Một lớp một khi được viết trong một gói có thể được sử dụng nhiều lần, không cần phải viết lại cùng một lớp.

#### Để sử dụng một gói, chúng tôi sử dụng từ khóa **import**.

import packageName.subPackageName.className;

#### Điều này sẽ nhập someClass từ một gói con chứa trong một gói.

#### Có hai loại gói:

1. Gói tích hợp
2. Gói do người dùng xác định

#### Hãy cùng khám phá cả hai.

### **Built-in packages**

#### Java cung cấp nhiều gói tích hợp mà chúng ta có thể sử dụng trực tiếp trong các chương trình của mình.

#### Trên thực tế, chúng tôi đã liên tục sử dụng chúng rất nhiều kể từ khi bắt đầu khóa học này.

#### Chúng ta hãy xem xét một vài gói được sử dụng thường xuyên và cách sử dụng chúng.

1. java.util - Gói này chứa các lớp liên quan đến thời gian, máy quét, thao tác mảng, cấu trúc dữ liệu, v.v.
2. java.io - Gói này chứa các lớp cần thiết cho các hoạt động đầu vào / đầu ra.
3. java.lang - Gói này chứa các lớp nền tảng cho việc thiết kế ngôn ngữ java như đối tượng.
4. java.awt - Gói này chứa các phần tử GUI phổ biến.

#### Nếu chúng ta muốn import bất kỳ class nào từ package util, chúng ta viết

import java.util.ClassName;

#### Trong trường hợp chúng ta muốn nhập tất cả các lớp của một gói, chúng ta viết

import java.util.\*;

#### Tuyệt vời! Bây giờ là lúc để tìm hiểu các gói do **người dùng** **xác định** là gì và cách chúng ta viết chúng. Vì vậy, hãy bắt đầu.

### **User-defined packages**

#### Các gói do chúng tôi tạo ra được gọi là gói do **người dùng** **xác định**.

#### Để tạo một gói, **gói** từ khóa được sử dụng.

#### Từ khóa gói theo sau là tên của **gói** hàng.

#### Tuyên bố này xuất hiện trên đầu tệp.

#### Các lớp được khai báo theo tuyên bố này trở thành một phần của gói.

#### Hãy viết một gói có tên là **phim hoạt hình**.

package cartoon; //a package named cartoon

#### Đó là nó.

#### Bây giờ, hãy viết một lớp **Tom** bên trong gói này.

package cartoon;

public class Tom{

//body

}

#### Tuyệt vời! Chúng tôi có **lớp Tom** bên trong **phim hoạt hình** gói.

#### Bây giờ, nếu chúng ta muốn truy cập vào lớp **Tom** có mặt trong **phim hoạt hình** gói, chúng ta có thể viết

import cartoon.Tom;

#### Và sau đó tiếp tục truy cập các phương thức và biến của lớp **Tom** như cách chúng ta thường làm.

### **Tóm tắt**

#### Một gói trong Java là một nhóm các loại lớp, giao diện và gói con tương tự.

#### Để sử dụng một gói, chúng tôi sử dụng từ khóa **import**.

import packageName.subPackageName.className;

#### Có hai loại **gói**: **Gói tích hợp** và Gói **do người dùng** **xác định**.

#### Để viết package s chúng ta viết **package packageName**;

### **What is an exception?**

#### Bạn sẽ làm gì khi nhìn thấy một cầu dao tốc độ trước mặt bạn?

#### Bạn áp dụng nghỉ giải lao, phải không?

#### Bộ ngắt tốc độ phá vỡ tốc độ bình thường của bạn.

#### Đây chính xác là những gì ngoại lệ làm cho chương trình của chúng tôi.

#### **Ngoại lệ** là một sự kiện làm gián đoạn luồng bình thường của chương trình.

#### Xử lý ngoại lệ cho phép chúng tôi xử lý lỗi thời gian chạy.

#### Do đó, nó duy trì luồng bình thường của chương trình.

#### Ngoại lệ có thể xảy ra do lỗi của người dùng, lỗi của lập trình viên hoặc do lỗi của bất kỳ tài nguyên vật lý nào.



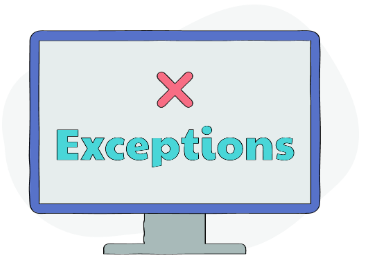
#### Có hai loại ngoại lệ:

1. Checked exception
2. Unchecked exception



#### Một ngoại lệ **được kiểm tra** là một ngoại lệ xảy ra tại thời điểm biên dịch, chúng còn được gọi là **ngoại lệ** thời gian biên dịch.

#### Lập trình viên cần xử lý những ngoại lệ này.



#### Một ngoại lệ **không được kiểm tra** là một ngoại lệ xảy ra tại thời điểm thực thi, chúng còn được gọi là **ngoại lệ** thời gian chạy.

#### Lỗi logic hoặc lỗi lập trình có ngoại lệ không được kiểm soát.

#### Lỗi không phải là ngoại lệ.

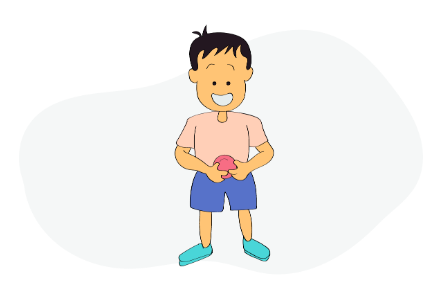
#### Trong lập trình, lỗi là thứ nằm ngoài tầm kiểm soát của người dùng hoặc lập trình viên và không thể xử lý như ngoại lệ.

### **Try, catch and finally**

#### Hãy thử, bắt và cuối cùng được sử dụng để xử lý các ngoại lệ trong Java.

#### Khối **try** được sử dụng để bao quanh khối mã có thể đưa ra ngoại lệ.

#### Khối **try** luôn được theo sau bởi **catch** hoặc **finally** là chặn.



#### Khối **catch** được sử dụng để xử lý ngoại lệ.

#### Nó luôn được sử dụng sau một khối **try**.

#### Có thể có nhiều khối **catch** theo sau là một khối **try**.

try{

//code

}

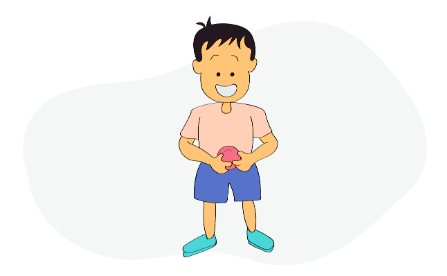
catch(exception\_name e){

//catch block

}

#### Đây là cách một khối **try catch** trông như thế nào.

#### exception\_name e chỉ ra ngoại lệ được ném nếu có.



#### Trong Java, **finally** khối luôn theo sau một khối thử hoặc bắt.

#### Đây là khối mã được thực thi bất kể việc thực thi có được xử lý hay không.

try{

//try block

}

finally{

//finally block

}

#### Đây là cách một **try finally** trông như thế nào.

#### Hãy viết một **try**, khối bắt để **catch** ngoại lệ được ném khi một số được chia cho 0.

class Main {

public static void main(String[] args) {

try{

int a = 10/0;

}

catch(Exception e){

System.out.println(e);

}

}

}

#### Đầu ra của đoạn trích trên sẽ là

java.lang.ArithmeticException: / bằng không

#### Đầu ra của đoạn trích cho biết đó là ngoại lệ được ném khi gặp ngoại lệ số học.java.lang.ArithmeticException

#### Và / **bằng** **0** là ngoại lệ đã xảy ra.

#### Hãy thêm một khối cuối cùng ở cuối mã.

class Main {

public static void main(String[] args) {

try{

int a = 10/0;

}

catch(Exception e){

System.out.println(e);

}

finally{

System.out.println("Division by zero.");

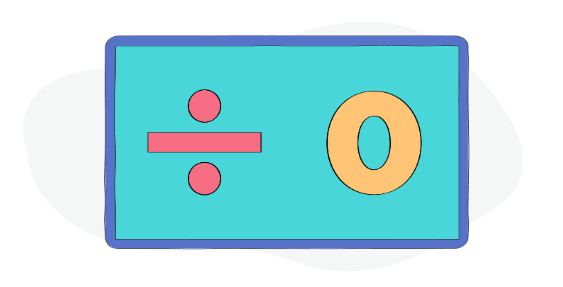
}

}

}

#### Đầu ra của đoạn trích trên sẽ là

java.lang.ArithmeticException: / bằng không Chia cho không.



#### Khối **finally** được thực thi và **Division** **by** **zero** được hiển thị bên dưới ngoại lệ bị bắt.

#### Đây là cách thử, bắt và cuối cùng là các khối hoạt động.

#### Hãy xem tại sao ném được sử dụng.

### **Throw**

#### Từ khóa ném được sử dụng để ném một ngoại lệ một cách rõ ràng.

### **Cú pháp của ngoại lệ ném java là**

throw new exception;

### **Hãy xem cách ném hoạt động.**

class Main {

static void div(int num){

if(num == 0)

throw new ArithmeticException("Division by 0.");

else

System.out.println(20/num);

}

public static void main(String[] args) {

div(0);

}

}

#### Đầu ra của đoạn trích trên sẽ là

Ngoại lệ trong luồng "main" java.lang.ArithmeticException: Chia cho 0.

#### Do đó, bằng cách sử dụng **ném**, chúng tôi đã ném một ngoại lệ tùy chỉnh cho biết **Chia** **cho** **0**.

#### Chúng tôi đã hoàn thành việc ném, bắt và sửa chữa mọi thứ cuối cùng.

#### Hãy đưa tất cả điều này vào thử nghiệm.

### **Tóm tắt**

#### Ngoại lệ là một sự kiện làm gián đoạn luồng bình thường của chương trình.

#### Xử lý ngoại lệ duy trì luồng bình thường của chương trình.

#### Khối **try** được sử dụng để bao quanh khối mã có thể đưa ra ngoại lệ.

#### Khối **catch** được sử dụng để xử lý ngoại lệ.

#### Khối **finally** luôn được thực thi bất kể ngoại lệ có được ném hay không.

#### Từ khóa ném được sử dụng để **throw** một ngoại lệ một cách rõ ràng.

### **File operations in java**

#### Java cho phép chúng ta thực hiện các thao tác trên **files**.

#### Chúng ta có thể đọc nội dung của một tệp và thậm chí ghi một số nội dung vào tệp thông qua chương trình Java.

#### Điều đó thật tuyệt vời phải không? :D

#### Để thực hiện các hoạt động tập tin, java cung cấp một số **class**.

#### Các lớp này đều được khai báo trong gói. java.io

#### Hãy khám phá các lớp này và các phương thức được sử dụng để thực hiện các thao tác trên tệp.

### **Creating a file**

#### Để thực hiện bất kỳ thao tác nào trên tệp, trước tiên chúng ta cần tạo tệp đó.

#### Để làm như vậy, phương pháp được sử dụng.createNewFile()

#### createNew**File**() phương thức được định nghĩa trong tập tin lớp có trong gói. java.io

#### createNewFile() Phương thức trả về giá trị boolean cho biết tệp đã được tạo thành công hay chưa.

#### Hãy viết một chương trình sẽ tạo một tệp có tên cho chúng ta.lazyMe.txt

import java.io.File; //importing File class from java.io package

public class Main {

public static void main(String[] args) {

File file = new File("lazyMe.txt"); //instantiating new object

try {

boolean newFile = file.createNewFile(); //create new file

System.out.println("Successful? "+newFile);

}

catch (Exception e) {

System.out.println("Oops! Some problem occured.");

}

}

}

#### Đầu ra của chương trình trên sẽ là:

Successful? true

#### Vì tệp đã được tạo thành công, phương thức trả về true.createNewFile()

#### Trong trường hợp tệp không được tạo thành công, đầu ra sẽ là

Successful? false

#### Tuyệt vời! Chúng tôi đã học cách tạo một tập tin. Hãy viết một số văn bản trong tập tin đó.

### **Writing into a file**

#### Để viết trong một tập tin java cung cấp nhiều lớp.

#### **FileWriter** - Nó có thể được sử dụng để ghi int và chuỗi vào tệp. Nó được ưa thích khi số lần ghi ít hơn.

#### **BufferedWriter** - Nó sử dụng một bộ đệm để ghi vào các tập tin. Nó được sử dụng trong trường hợp số lần ghi nhiều hơn. Nó cải thiện hiệu suất.

#### **FileOutputStream - FileWriter** và BufferedWriter được sử dụng để ghi văn bản vào tệp. FileOutputStream được sử dụng để ghi dữ liệu thô vào các tệp.

#### Hãy viết một chương trình bằng java để viết một chuỗi vào một tệp bằng cách sử dụng **FileOutputStream**.

import java.io.FileOutputStream;

import java.io.File;

import java.io.IOException;

public class Main {

public static void main(String[] args) {

FileOutputStream writeFile = null;

File file;

String myText = "Let’s write something in this file.";

try {

file = new File("lazyMe.txt"); //new File object

writeFile = new FileOutputStream(file); //FileOutputstream object

byte[] byteData = myText.getBytes(); //string to bytes conversion

writeFile.write(byteData); //writing converted byteData to file

System.out.println("Successful.");

}

catch (IOException ioe) {

System.out.println("OOPs! There was some problem.");

}

}

}

#### Đầu ra của chương trình sẽ là

Thành công.

#### Và "**Let's write** **something in** **this** file." sẽ được viết trong **file** lazyMe.txt.

#### Ở đây, chúng tôi đã sử dụng **FileOutputStream** và để viết văn bản trong tệp.write()

### **Reading a file**

#### Nội dung của tệp có thể được đọc bằng **FileInputStream** và **BufferedReader**.

#### read() Phương thức được sử dụng để đọc nội dung của ký tự tệp theo ký tự.

#### Hãy đọc nội dung của tệp bằng **FileInputStream**.

import java.io.FileInputStream;

public class Main {

public static void main(String args[]){

try{

FileInputStream myFile= new FileInputStream("lazyMe.txt");

int i=0;

while((i=myFile.read())!=-1){

//reading every character

System.out.print((char)i); // printing to the screen

}

myFile.close(); //closes the file after reading

}

catch(Exception e){

System.out.println(e);

}

}

}

#### Nội dung của tệp sẽ được đọc và in ra màn hình.

#### Trong trường hợp không tìm thấy tệp

java.io.FileNotFoundException: lazyMe.txt (Không có tệp hoặc thư mục như vậy)

#### Sẽ được in trên màn hình đầu ra.

### **Delete a file**

#### Bây giờ, chúng ta biết cách tạo, đọc và viết một tệp.

#### Hãy xem làm thế nào chúng ta có thể xóa một tập tin.

#### delete() phương thức của lớp **File** được sử dụng để làm như vậy.

#### Đây là một chương trình trong java mà xóa một tập tin.

import java.io.File;

public class Main {

public static void main(String[] args) {

File file = new File("lazyMe.txt"); //file to be deleted

boolean deleteMe = file.delete(); //delete method

System.out.println("Deleted? " + deleteMe);

}

}

#### delete() Phương thức trả về giá trị boolean dựa trên việc tệp đã bị xóa hay chưa.

#### Vì vậy, đầu ra sẽ là

Xóa? thật

#### Và, nếu tệp không bị xóa

Xóa? sai

### **Tóm tắt**

#### Để thực hiện các hoạt động tệp, java cung cấp một số lớp được khai báo trong gói.java.io

#### createNewFile() Phương thức được sử dụng để tạo một tệp mới.

#### write() phương thức được sử dụng để ghi vào một tệp và bất kỳ FileWriter, **BufferedWriter** hoặc **FileOutputStream** nào cũng có thể được sử dụng để làm như vậy.

#### Tương tự, nội dung của tệp có thể được đọc bằng **FileInputStream** và **BufferedReader**. Phương pháp được sử dụng để làm như vậy.read()

#### Và, phương thức của lớp **File** được sử dụng để xóa một tệp trong Java.delete()