Khách Inbound và Outbound

User tách riêng 1 DB

Db dùng: Redis hoặc kafka

* Buid trên docker

Mọi thứ đều đi qua Gateway

-header

+ osaptype

+ osapversion

Bản tin trả ra:

+ Http Status

+ Errorcode: “1000” do mk tạo ra

+ message

+ data

Phase1: 2 month: hình thành luồng cơ bản

Sprint 1:

* User: xem tour, đặt tour, thanh toán, rating …
* Agency: đăng ký, đăng tour, quản lý tour – lịch tour/ ….
* Admin: quản trị user/agency

Phase 2: nâng cấp, mobile ….

Quản lý: aglle / trello…

Mỗi ngày nói ra được:

+ done

+ problem

+ plan

# .Primitive & Object datatype

Kiểu dữ liệu là dùng để lưu trữ thông tin, giá trị mà chương trình biên dịch ra

1. Thế nào là kiểu dữ liệu nguyên thủy?

* Kiểu dữ liệu nguyên thủy là kiểu dữ liệu được chương trình cung cấp và hỗ trợ sẵn, nó được dùng để lưu trữ các giá trị đơn giản
* Kiểu dữ liệu nguyên thủy là tham trị

Có 2 kiểu chính là kiểu boolen và kiểu số

Kiểu logic: boolean

Kiểu số:

+ Kiểu nguyên:

+ Kiểu số thực: Float, Double

Mặc định Java xem kiểu số nguyên là int và số thực là double, nên khi ta khai báo kiểu dữ liệu float thì them f hoặc F, long thì thêm l hoặc L vào cuối số. Với kiểu float thì chắc chắn phải thêm f (hoặc F) vào cuối số vì nó có phạm vi nhỏ hơn mặc định double. Còn long thì giả sử nó vượt quá phạm vi của int thì them l (hoặc L) ở cuối

Overflow (tràn ra): đó là khi giá trị vượt quá mức cho phép mà chương trình cho ra, giả sử long là kiểu dữ liệu cho phép lớn nhất rồi mà có phép cộng, cộng thêm 1 đơn vị nữa chẳng hạn nó sẽ vượt quá mức cho phép của long thì lúc này Java sẽ đảo ngược chiều về hàng âm nhỏ nhất và bắt đầu đếm lại từ đó

Underflow (tràn xuống): ngược lại với overflow, nghĩa là khi nhỏ quá vượt mức lưu trữ, phạm vi của chương trình thì nó sẽ xoay chiều sang số dương lớn nhất và giảm xuống từ đó

1. Thế nào là kiểu dữ liệu đối tượng

* Kiểu dữ liệu đối tượng là kiểu dữ liệu mang tính chất tham chiếu, biến của nó sẽ tham chiếu đến đối tượng. Biến của nó sẽ không lấy giá trị mà lấy địa chỉ bộ nhớ của đối tượng đó
* Trong java có 3 kiểu dữ liệu đối tượng:

+ Array: đó là 1 mảng động chứa các phần tử có cùng kiểu dữ liệu

+ Class: do người dùng tự định nghĩa

+ interface: dữ liệu kiểu lớp giao tiếp do người dùng tự định nghĩa

1. Chuyển đổi kiểu dữ liệu giữa 2 dạng này như thế nào ?

* Thông qua Wrapper class, cụ thể là autoboxing và unboxing

Java cung cấp 1 Wrapper class bao gồm các đối tượng Integer, Double, Long, Boolean …

nó sẽ tự động chuyển đổi 2 kiểu dữ liệu đó lẫn nhau

Integer a = Integer.valueOf(9); // boxing

Integer a = 9; // autoboxing

int b = a.intValue(); // unboxing

int c = a; // auto unboxing

* Chuyển đổi thủ công

int a = 10;

Integer b = Integer.valueOf(a);

Interger c = new Integer(10);

int d = c.intValue;

Mỗi kiểu điều có ưu nhược điểm riêng

* Kiểu nguyên thủy:

+ Ưu điểm: tốn ít bộ nhớ, hiệu suất tốt hơn, đoạn mã code nhìn rõ ràng

+ Nhược điểm: cứng nhắc, không có tính linh hoạt, không thể tùy biến được

* Kiểu đối tượng:

+Ưu điểm: đa dạng, linh hoạt, dễ tùy biến, tái sử dụng

+ Nhược điểm: tốn bộ nhớ khi ta không sử dụng hợp lý

Autoboxing: là quá trình tự biến đổi kiểu dữ liệu đối tượng sang kiểu dữ liêu tham chiếu

Ví dụ: Integer = 2; // autoboxing

Integer a = new Integer(2)

Integer a = Interger.valueOf(2)

Unboxing: là quá trình biến đổi từ kiểu dữ liệu đối tượng sang nguyên thủy

Integer b = 2;

int c = b.intValue()

# Java-OOP-1

Hướng đối tượng là mô hình hóa hệ thống trong cuộc sống trong phần mềm dựa trên các đối tượng

Các tính chất hướng đối tượng:

* Tính đóng gói: gói gém và che giấu dữ liệu, bảo mật dữ liệu trong đối tượng, và chỉ được truy cập dữ liệu bên trong đối tượng qua các phương thức và không cho phép gọi trực tiếp chúng

Thể hiện qua access modifiers, khi 1 biến hoặc 1 phương thức khai báo dưới dạng:

+ public(công khai): thì chúng ta có thể truy cập chúng ở khắp mọi nơi

+ private(riêng tư): chỉ được gọi trong class mà chúng được định nghĩa

+ defaut: được gọi trong cùng 1 package

+ protected: được truy cập trong cùng 1 package, và các lớp con của nó

public class Animal {  
 private String name;  
  
 public Animal(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

như trên ta thấy khi khởi tạo đối tượng Animal, khi xét các thuộc tính ta phải gọi thông qua phương thức trung gian đó là setter và getter. 2 phương thức đó sẽ kiểm soát được giá trị mà ta muốn đưa vào xem có hợp lệ không

* Tính kế thừa: khi một lớp con kế thừa 1 lớp cha thì nó sẽ được thừa kế toàn bộ những phương thức và thuộc tính của lớp cha. Nhưng nếu lớp cha đặt trạng thái ở private thì nó sẽ không kế thừa được trừ khi trong cùng 1 class
* Tính đa hình: trên cùng 1 phương thức nhưng mỗi đối tượng khác nhau thì nó là khác nhau, điều này trong java thường thể hiện qua:

+ override(ghi đè): khi 1 lớp con kế thừa 1 lớp cha, mà nó không muốn kế thừa hành động của của lớp cha thì nó sẽ ghi đè nội dung lên theo ý của nó

+ overrloading(nạp chồng): là ta có thể định nghĩa nhiều phương thức có cùng tên nhưng khác kiểu dữ liệu, kiểu trả về và số lượng đối số truyền vào

public class Overload {  
  
 public int tinhTong(int a, int b) {  
 return a+b;  
 }  
 public double tinhTong(double a, double b){  
 return a+b;  
 }  
   
 public double tinhTong(double a, double b, double c){  
 return a+b+c;  
 }  
  
 public static void main(String[] args) {  
 Overload overload = new Overload();  
  
 System.*out*.println(overload.tinhTong(3,2,1));  
 }  
}

+ Sử dụng đối tượng đa hình (Polymorphic Object): biến của lớp cha có thể tham chiếu đến đối tượng của lớp con

* Tính trừu tượng: cho phép chúng ta tạo ra 1 lớp mà không cần cung cấp cách triển khai trong phương thức, nó giúp tạo ra 1 cái tiêu chuẩn, 1 mô tả mà không cần cung cấp nội dung, để khi các lớp con kế thừa nó thì sẽ phải triển khai tuân theo nguyên tắc của mô tả đó và triển khai phương thức bên trong

Phân biệt giữa lớp và đối tượng trong Java

* Lớp là 1 bản thiết kế hoặc như 1 bản mô tả đến đối tượng, nó cung cấp các thuộc tính và hành vi để định nghĩa lên đối tượng
* Đối tượng là 1 thực thể cụ thể của 1 lớp, có hành vi và được tạo ra dựa trên thiết kế của lớp
* Như vậy ta có thể hiểu đối tượng được tạo ra dựa trên class, đối tượng là 1 thực thể còn class là 1 logic được viết ra, trong java khi runtime thì nó sẽ không cấp vùng nhớ cho class mà cấp cho đối tượng được tạo ra từ class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Trong class | Trong package | Ngoài package không thuộc class con | Ngoài package thuộc class con | World |
| private | Y | X | X | X | X |
| default | Y | Y | X | X | X |
| protected | Y | Y | X | Y | X |
| public | Y | Y | Y | Y | Y |

# 2. Static & final

Thế nào là static trong Java?

* Khi 1 static được khai báo nghĩa là nó thuộc của lớp đó chứ không thuộc về đối tượng cụ thể nào của lớp đó. Các đối tượng sẽ dùng chung những gì khai báo static

Ý nghĩa từ khóa static trong Java?

* Static khi khai báo biến thì biến này sẽ được coi là của lớp, mỗi đối tượng sẽ không phải khởi tạo nó và dùng luôn nó, các đối tượng sẽ chia sẻ giá trị của biến được gán static này

Ưu điểm:

+ chia sẻ giá trị cho nhau giữa các đối tượng

+ tiết kiệm bộ nhớ

+ kết hợp với từ khóa final để khai báo hằng

+ theo dõi được đối tượng tạo ra

* Static trong phương thức thì chỉ có class tạo ra nó mới được sử dụng, nó không phụ thuộc vào đối tượng được tạo ra hay không và các đối tượng cũng không được dùng nó. Nó không phụ thuộc vào tham số truyền vào của đối tượng

Ưu điểm

+ Thực hiện các hành động không liên quan đến đối tượng, nhưng vẫn liên quan đến lớp

+ Tạo các tiện ích chung như là: tính tổng, max, min, trung bình ….

+ Xây dựng phương thức main, vì phương thức main được chạy ngay từ đầu khi bắt đầu câu lệnh, thì với static nó không cần khởi tạo đối tượng nên JVM không cần phải khởi tạo đối tượng cho class chứa nó

+ Tạo ra các factory method, đây là các phương thức tạo ra các đối tượng mới dựa trên tham số truyền vào

* Chúng ta có thể ghi đè 1 hàm là static hoặc private hay không?

Không thể ghi đè static hoặc private

+ static: vì chúng thuộc của lớp không phải của đối tượng. Bình thường khi kế thừa, lớp con thừa hưởng của lớp cha và có thể ghi đè, nó dành cho đối tượng

+ private: do phạm vi của nó chỉ được truy cập khi trong cùng 1 class nên không thể ghi đè được

Có thể truy cập biến non-static trong 1 lớp static không

* Truy cập trực tiếp thì không, còn gián tiếp thì có. Dùng biến tham chiếu của đối tượng, bằng cách ta phải khởi tạo đối tượng của lớp đó và dùng biến tham chiếu đó gọi ra

Thế nào là final trong Java? Có thể kế thừa được phương thức final không?

* Final trong java có những loại sau:

+ Biến final khi được khai báo sẽ có giá trị không đổi

+ Phương thức final: khi lớp con kế thừa phương thức này thì nó chỉ được kế thừa chứ không được ghi đè chúng

+ class final: thì không thể kế thừa

+ biến static final trống: được gọi là biến static final trống, sẽ được khai báo sau trong block static {}

* Có thể kế thừa phương thức final nhưng không thể ghi đè chúng vì khi 1 phương thức được khai báo là final nó sẽ không đổi, nên lớp con kế thừa nó chỉ được phép dùng chứ không được khi đè

Có thể khi báo biến final không khởi tạo giá trị ban đầu tại thời điểm khai báo không?

* Có thể không khai báo, nhưng phải thông qua hàm constructor hoặc với static final thì ta có thể khai báo sau qua khối block static

Thế nào là tham số final?

* Khi 1 tham số final được khai báo trong 1 phương thức hay 1 constructor, thì nó sẽ không được gán giá trị trong phương thức hay constructor đó. Mà nó chỉ nhận giá trị khi 1 đối tượng gọi đến hàm đó và truyền đối số vào
* Khi một final khai báo với 1 collection thì có thể hiểu là biến tham chiếu đến collection đó không được thay đổi, size và giá trị của collection vẫn thay đổi được nhưng biến tham chiếu đến nó không được cho giá trị khác

# 3. This & Super

Thế nào là **this** và **super** trong java? 2 từ khóa trên được sử dụng trong các trường hợp nào?

* this nó tham chiếu đến lớp hiện tại như các biến istance, hàm, constructor, trả về instance của lớp hiện tại, được truyền như 1 tham số trong phương thức và constructor

ví dụ: + this tham chiếu tới biến instance trong lớp hiện tại

public class Test {  
  
 private String name;  
 private int age;  
  
 public Test(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public void run(){  
 System.*out*.println("Name: " + this.name);  
 System.*out*.println("Age: " + this.age);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.run();  
 }  
  
}

**+** gọi đến phương thức đến lớp hiện tại

public class Test {  
  
 private String name;  
 private int age;  
  
 public Test(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 System.*out*.println("Welcome ");  
 this.run();  
 }  
  
 public void run(){  
 System.*out*.println("Name: " + this.name);  
 System.*out*.println("Age: " + this.age);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.printWebsite();  
 }  
  
}

+ gọi đến constructor của lớp hiện tại. Khi bạn có nhiều constructor và bạn muốn sử dụng lại

public class Test {  
 private int id;  
 private String name;  
 private int age;  
  
 public Test() {  
 System.*out*.println("Ok");  
 this.id = 1;  
 this.name = "Nam";  
 }  
  
 public Test(String name) {  
 this();  
 this.name = name;  
 }  
  
 public Test(String name, int age){  
 this(name);  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 System.*out*.println("Welcome ");  
 this.run();  
 }  
  
 public void run(){  
 System.*out*.println("Id: " + this.id);  
 System.*out*.println("Name: " + this.name);  
 System.*out*.println("Age: " + this.age);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.printWebsite();  
 }  
  
}

+ trả về instance cho lớp hiện tại

public class Test {  
 private int id;  
 private String name;  
 private int age;  
   
 public Test(String name, int age){  
 this.name = name;  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 System.*out*.println("Welcome ");  
 this.run();  
 }  
  
 public void run(){  
 System.*out*.println("Id: " + this.id);  
 System.*out*.println("Name: " + this.name);  
 System.*out*.println("Age: " + this.age);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.printWebsite();  
 }  
  
 public Test setNameTest(String name) {  
 this.name = name;  
 return this;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

+ cũng có thể dùng this truyền vào như 1 tham số

public class Test {  
 private int id;  
 private String name;  
 private int age;  
  
 public Test(String name, int age){  
 this.name = name;  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 System.*out*.println("Welcome ");  
 this.run();  
 }  
  
 public void run(){  
 TestCon testCon = new TestCon();  
 testCon.getAll(this);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.run();  
 }  
  
 public Test setNameTest(String name) {  
 this.name = name;  
 return this;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
}  
  
class TestCon {  
  
 public void getAll(Test test){  
 System.*out*.println("Name: " + test.getName());  
 System.*out*.println("Age: " + test.getAge());  
 }  
}

+ Được truyền như 1 tham số trong constructor

public class Test {  
 private int id;  
 private String name;  
 private int age;  
  
 public Test(String name, int age){  
 this.name = name;  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 TestCon testCon = new TestCon(this);  
 testCon.print();  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.printWebsite();  
 }  
  
 public Test setNameTest(String name) {  
 this.name = name;  
 return this;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
}  
  
class TestCon {  
 private Test test;  
 public TestCon(Test test) {  
 this.test = test;  
 }  
  
 public void print(){  
 System.*out*.println("Name: " + test.getName());  
 System.*out*.println("Age: " + test.getAge());  
 }  
}

Nói tóm lại This sẽ có những mục đích như sau:

this thực ra nó là 1 biến tham chiếu và nó được sử dụng để tham chiếu đến đối tượng hiện tại của lớp đó

+ tham chiếu đến biến instance của lớp hợp tại ( this.age = age )

+ gọi đến phương thức hoặc constructor của lớp hiện tại ( this.run() hoặc this(), this(name) )

+ có thể trả về 1 instance (return this;)

+ được truyền vào như 1 tham số trong phương thức hoặc construtor ( testCon.getAll(this), Testcon testcon = new Testcon(this) )

* super: là 1 biến tham chiếu nó được sử dụng để tham chiếu đến đối tượng cha gần nhất của lớp con
* Bất cứ khi nào 1 instance lớp con tạo ra sẽ có 1 intance của lớp cha được tạo ra ngầm định, nó được gọi là biến super.
* Nó sẽ gọi constructor, thuộc tính và phương thức của lớp cha gần nhất

+ goi đến constructor của lớp cha gần nhất của nó:

class Parents {  
 private String name;  
  
 public Parents() {  
 System.*out*.println("This is parents");  
 print();  
 }  
 public Parents(String name){  
 this.name = name;  
 System.*out*.println("This is parents");  
 print();  
 }  
 public void print(){  
 System.*out*.println("Hello world: " + name);  
 }  
}  
  
class Child extends Parents {  
 public Child(){  
 System.*out*.println("This is Child");  
 }  
 public Child(String name){  
 super(name);  
 System.*out*.println("This is Child");  
 }  
  
 public static void main(String[] args) {  
 Child child1 = new Child();  
 System.*out*.println("---");  
 Child child2 = new Child("Hai");  
 }  
}

+ gọi đến thuộc tính (field) cha gần nhất

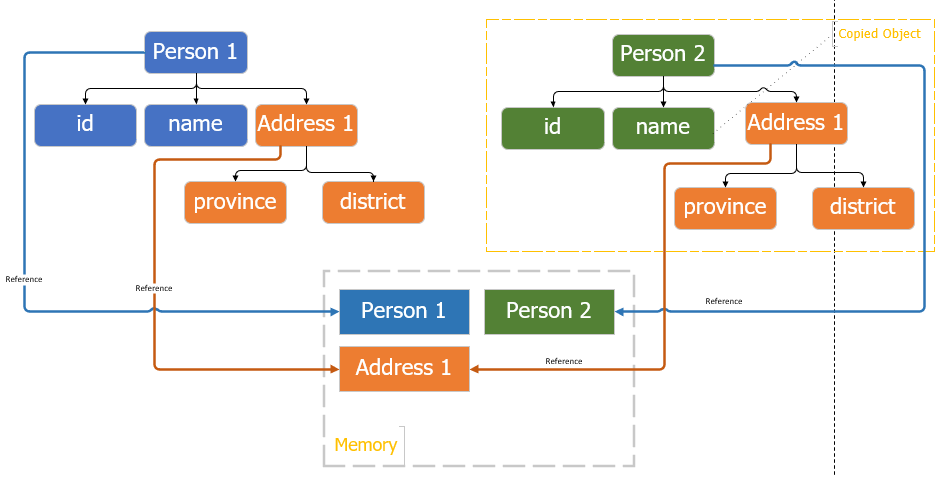
Lưu ý: nếu như thuộc tính của lớp con khác tên thuộc tính của lớp cha thì không cần dùng từ super

+ gọi đến phương thức của lớp cha gần nhất. Giải quyết trường hợp ghi đè, và trong trường hợp bạn vẫn gọi hàm từ lớp cha ta sẽ dùng super

**Phân biệt Reference Copy (copy kiểu tham chiếu) và Object Copy (copy toàn bộ đối tượng ) ? Nêu ví dụ cụ thể**

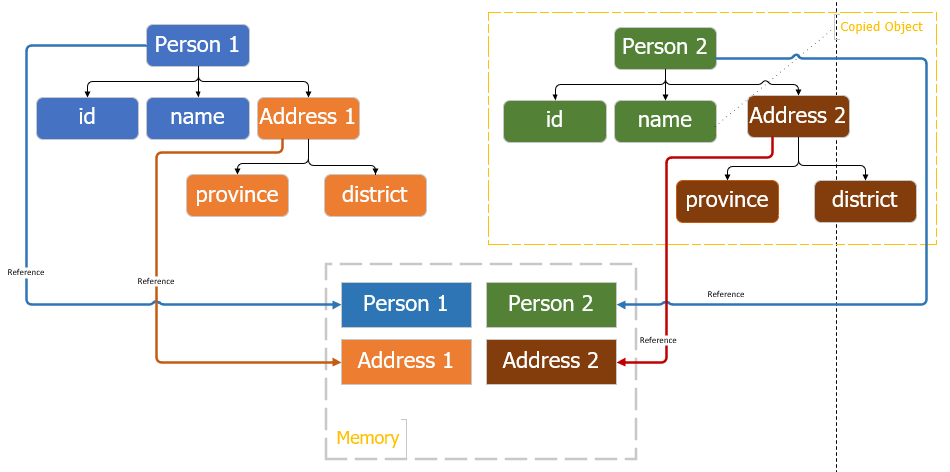
Trong Java có 2 kiểu copy đó là copy kiểu tham chiếu và copy đối tượng

* Reference Copy (copy tham chiếu): khi chúng ta thay đổi thuộc tính bên trong 1 đối tượng thì đối tượng copy đối tượng đố qua toán tử ‘=’ thì đối tượng copy đó cũng sẽ thay đổi theo. Vì bản chất đây là copy địa chỉ chứ không phải giá trị, 2 đối tượng này sẽ cùng có 1 địa chỉ
* public class Student {  
   int id;  
   String name;  
    
   public Student() {  
   }  
    
   public Student(int id, String name) {  
   this.id = id;  
   this.name = name;  
   }  
    
   public void show(){  
   System.*out*.println("Id: " + id);  
   System.*out*.println("Name: " + name);  
   }  
  }  
    
  class Test {  
   public static void main(String[] args) {  
   Student s1 = new Student(1, "Hai");  
   Student s2 = s1;  
   s1.show();  
   s2.show();  
    
   System.*out*.println("After changed: ");  
   s1.id = 4;  
   s1.name = "Nam";  
   s1.show();  
   s2.show();  
    
   }  
  }
* Object copy: khi đối tượng 1 copy đối tượng 2 thì khi object 2 thay đổi giá trị thuộc tính thì đối tượng 1 sẽ không ảnh hưởng, thực ra là 2 đối tượng cùng tạo trên 1 class
* Shallow copy/ Shallow cloning: đối tượng copy sẽ copy những thuộc tính kiểu dữ liệu nguyên thủy (int, float, …) và String, nhưng các thuộc tính thuộc dạng Object vẫn tham chiếu tới địa chỉ cũ. Nghĩa là khi thuộc tính Object thuộc đối tượng gốc thay đổi thì nhưng đối tượng copy nó sẽ thay đổi theo



class Address {  
 String name;  
  
 public Address() {  
 }  
  
 public Address(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 return "Address{" +  
 "name='" + name + '\'' +  
 '}';  
 }  
}  
public class Person implements Cloneable{  
 int id;  
 String name;  
 Address address;  
  
 public Person(int id, String name, Address address) {  
 this.id = id;  
 this.name = name;  
 this.address = address;  
 }  
  
 @Override  
 protected Person clone() throws CloneNotSupportedException {  
 return (Person) super.clone();  
 }  
  
 @Override  
 public String toString() {  
 return "Person{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", address=" + address +  
 '}';  
 }  
}  
  
class TestCopy {  
 public static void main(String[] args) throws CloneNotSupportedException{  
 Address address = new Address("Hải Phòng");  
 Person person = new Person(1, "Hai", address);  
 Person personCopy = person.clone();  
 System.*out*.println(person);  
 System.*out*.println(personCopy);  
  
 System.*out*.println("----------------");  
  
 person.id = 2;  
 person.name = "Duy";  
 person.address.name = "Hà nội";  
 System.*out*.println(person);  
 System.*out*.println(personCopy);  
 }  
}

* Deep cloning



class Address implements Cloneable{  
 String name;  
  
 public Address() {  
 }  
  
 public Address(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 return "Address{" +  
 "name='" + name + '\'' +  
 '}';  
 }  
  
 @Override  
 protected Address clone() throws CloneNotSupportedException {  
 return(Address) super.clone();  
 }  
}  
public class Person implements Cloneable{  
 int id;  
 String name;  
 Address address;  
  
 public Person(int id, String name, Address address) {  
 this.id = id;  
 this.name = name;  
 this.address = address;  
 }  
  
 @Override  
 protected Person clone() throws CloneNotSupportedException {  
 Person person = (Person) super.clone();  
 person.address = this.address.clone();  
 return person;  
 }  
  
 @Override  
 public String toString() {  
 return "Person{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", address=" + address +  
 '}';  
 }  
}  
  
class TestCopy {  
 public static void main(String[] args) throws CloneNotSupportedException{  
 Address address = new Address("Hải Phòng");  
 Person person = new Person(1, "Hai", address);  
 Person personCopy = person.clone();  
 System.*out*.println(person);  
 System.*out*.println(personCopy);  
  
 System.*out*.println("----------------");  
  
 person.id = 2;  
 person.name = "Duy";  
 person.address.name = "Hà nội";  
 System.*out*.println(person);  
 System.*out*.println(personCopy);  
 }  
}

**Bạn hiểu gì về từ khóa instance of trong Java**

Biến là 1 vị trí trong bộ nhớ dùng để lưu trữ dữ liệu

Integer a = 1;

+ Instance dùng để kiểm tra 1 đối tượng có thể hiện của 1 kiểu dữ liệu cụ thể hay không

+ Trả về True khi nó đúng

+ Với biến có giá trị null thì nó trả về false

# 4 . String

Thế nào là mutable và immutable object trong java?

* Mutable object là có thể thay đổi trạng thái của đối tượng
* Immutable object là khi ta khởi tạo 1 object thì không thể thay đổi trạng thái của object này ( có thể hiểu là chỉ có thể get chứ không thể nào set giá trị cho nó)

Phân biệt String, String buider và String buffer?

* String là immutable còn String buider và String buffer là immutable
* String Buffer và String Buider là giống nhau chỉ khác 1 điều đó là String Buider dành cho 1 luồng còn String Buffer dành cho đa luồng

Khi nào sử dụng String, String buider và String buffer?

* String có tính chất là immtable nên cần gán cho những giá trị không thay đổi hoặc ít thay đổi để nó đỡ phải tạo ra bản sao khác do tính chất của nó

Hiệu năng giữa 3 đối tượng trên

public class main {  
 static long *j* = 100000l;  
 public static void main(String[] args) {  
 Long start1 = System.*nanoTime*();  
 *stringConcat*();  
 Long end1 = System.*nanoTime*();  
 System.*out*.println("String: " + (end1-start1));  
  
 Long start2 = System.*nanoTime*();  
 *stringBuider*();  
 Long end2 = System.*nanoTime*();  
 System.*out*.println("String Buider: " + (end2-start2));  
  
 Long start3 = System.*nanoTime*();  
 *stringBuffer*();  
 Long end3 = System.*nanoTime*();  
 System.*out*.println("String Buider: " + (end3-start3));  
  
 }  
  
 public static String stringConcat() {  
 String res = "";  
 for (int i = 0; i < *j*; i++) {  
 res+=i;  
 }  
 return res;  
 }  
  
 public static StringBuilder stringBuider() {  
 StringBuilder res = new StringBuilder();  
 for (int i = 0; i < *j*; i++) {  
 res.append(i);  
 }  
 return res;  
 }  
  
 public static StringBuffer stringBuffer() {  
 StringBuffer res = new StringBuffer();  
 for (int i = 0; i < *j*; i++) {  
 res.append(i);  
 }  
 return res;  
 }  
  
}

String: 2572825100

String Buider: 3529800

String Buider: 3409400

* String pool ra đời để giải quyết vấn đề bộ nhớ cho String, ví dụ String a = “hello world” chiếm 11 ký tự tương ứng với 22 byte (trong java) trong khi đó với kiểu số lớn nhất như long mới chiếm có 8byte. Vậy String pool ra đời, nó chứa những chuỗi ký tự mà giả sử có 2 đối tượng có cùng 1 chuỗi thay vì mỗi cái tạo ra 1 object mới và trỏ đến thì 2 đối tượng đó đều trỏ đến chuỗi được lưu trong string pool. Vậy nên String java mới để là immutable
* Tuy nhiên để giải quyết bài toán cộng chuỗi thì với String sẽ tốn rất nhiều bước, khi 2 chuỗi cộng lại với nhau ta cần tìm ra 1 vùng đất mới để lưu vừa 2 chuỗi này, đồng thời sao lưu 2 chuỗi đó và copy đến vùng đất mới được tìm, rồi bộ dọn rác kiểm tra xem 2 chuỗi vừa sao lưu còn dùng nữa hay không để dọn.
* Vậy nên java sinh ra String buider và String buffer
* Tóm lại String dùng khi dữ liệu không thay đổi hoặc ít thay đổi, còn String buider và String buffer dùng khi nội dung hay thay đổi

# 5. Memory

Phân biệt JVM, JRE, JDK?

* JVM là tạo ra môi trường máy ảo phụ trách biên dịch bytecode
* JRE: bao gồm JVM và 1 số thư viện và file để JVM sử dụng lúc thực thi
* JDK: là bộ cài bao gồm JRE + công cụ phát triển. Trong đó có javac dùng để biên dịch các file .java thành .class (byte code) rồi đưa chúng cho JVM biên dịch thành mã máy

Thế nào là cấp phát tĩnh, cấp phát động?

* Điểm chung là cả đều được cấp phát lúc runtime
* Cấp phát tĩnh là kiểu cấp phát đã được xác định kích thước cần sử dụng trước khi biên dịch
* Cấp phát động là kiểu cấp không biết trước được kích thước từ trước, nó cho phép quản lý phân bổ vùng nhớ trong khi chạy chương trình

Phân biệt bộ nhớ Heap và Stack?

* Bộ nhớ Stack chứa những kiểu dữ liệu nguyên thủy, các phương thức, tham số truyền vào và biến tham chiếu đến đối tượng. Nó được sử dụng để cấp phát bộ nhớ tĩnh và thực thi một luồng
* Không gian Heap được sử dụng để cấp phát bộ nhớ động và các lớp JRE khi chạy. Nó chứa các đối tượng (khi thực thi toán tử new)

# 6. Interface & Abstract class

Thế nào là interface và abstract class?

* abstract class nó cũng giống như 1 class thông thường đều có các phương thức và thuộc tính, nhưng nó có thêm cái là có abstract method.

+ abstract method này thì chỉ được khai báo và không nó nội dung, khi các lớp con kế thừa lớp cha có phương thức abstract method thì chúng bắt buộc phải ghi đè nó

+ abstract class không thể tạo ra đối tượng trực tiếp từ nó mà phải thông qua lớp con

* interface thì nó chứa các abstract method và mặc định của nó là public, thuộc tính của nó mặc định là final static
* từ java8 và 9 trở đi thì các method trong interface có thể triển khai nội dung với các từ khóa như là default, static, private
* default thì lớp impl có thể ghi đè nội dung hoặc sử dụng luôn cũng được
* lớp static thì nó thuộc về lớp nên ko thể ghi đè
* private method thì chỉ lưu hành nội bộ trong interface đó
* interface có thể đa kế thừa do nó nguyên lý nó bắt các lớp con khi kế thừa phải ghi đè nó

Abstract class được dùng khi nó triển khai nội dung 1 số chức năng dùng chung của hệ thống, các chức năng còn lại để cho các extends phải hoàn thành

Interface là khi bạn tạo 1 bộ khung chuẩn để các lớp con impl phải thực hiện nó

Thế nào là Upcasting và Downcasting trong java?

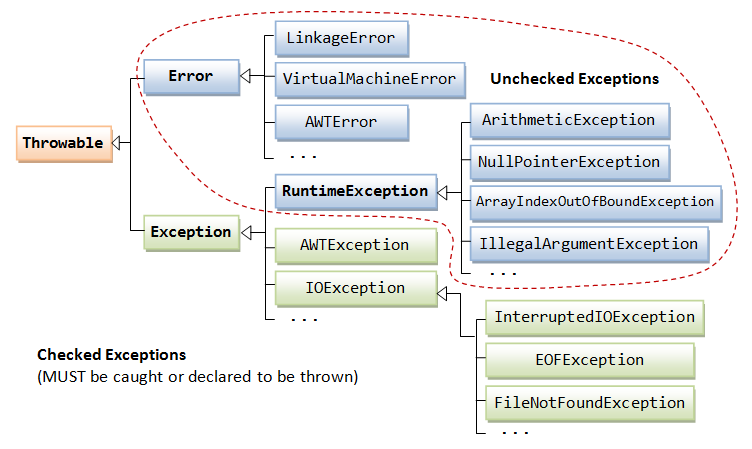
* Cả 2 đều là ép kiểu với những đối tượng có mối quan hệ kế thừa
* Khi biến tham chiếu của lớp cha chiếu đến đối tượng của lớp con thì đó là Upcasting
* public class Upcasting {  
   public static void main(String[] args) {  
   Cat cat = new Cat();  
   Animal animal1 = cat; // Chuyển kiểu không tường minh  
   Animal animal2 = (Cat) cat; // chuyển kiểu tường minh  
    
   cat.eat();  
   cat.meow();  
   animal1.eat();  
   animal2.eat();  
   }  
  }
* Khi biến tham chiếu của lớp con chiếu đến đối tượng của lớp cha thì đó là downcasting

public class Downcasting {  
 public static void main(String[] args) {  
 Animal animal = new Cat();  
 Cat cat = (Cat) animal; // downcasting  
 cat.meow();  
 cat.eat();  
 }  
}

* Tại sao lại phải dùng đến Upcasting? Bởi vì upcasting được sử dụng nhiểu trong tính đa hình của lập trình hướng đối tượng, khi ta khai báo 1 biến có kiếu dữ liệu là lớp cha thì ta có thể tham chiếu đến tất cả các lớp con của nó, đỡ phải khai báo kiểu dữ liệu cụ thể
* List<Animal> list = new ArrayList<>();  
  Animal cat = new Cat();  
  Animal dog = new Dog();  
  list.add(cat);  
  list.add(dog);  
  for (Animal item: list  
   ) {  
   item.eat();  
  }
* Tại sao phải dùng đến Downcasting? Khi ta cần gọi ra các phương thức mà chỉ trong lớp con mới có mà lớp cha không có
* List<Animal> list = new ArrayList<>();  
   list.add(new Dog());  
   list.add(new Cat());  
    
   for (Animal item: list  
   ) {  
   item.eat();  
   if (item instanceof Dog){  
   Dog dog = (Dog) item;  
   dog.gow();  
   } else if (item instanceof Cat){  
   Cat cat = (Cat) item;  
   ((Cat) item).meow();  
   }  
   }  
  }

# 7. Exception

Thế nào là **check** và **uncheck** expception trong Java?



* Exption là một sự kiện xảy ra trong quá trình thực thi 1 chương trình trong java, nó làm phá vỡ luồng đang chạy của chương trình, thậm chí là chết cả chương trình.
* Điểm khác biệt giữa 2 lớp checked và unchecked là thời điểm kiểm tra expception
* Check expception là loại expception xảy ra trong lúc compile time và buộc ta phải handle nó
* public class Test {  
   public static void main(String[] args) {  
   BufferedReader br = null;  
   try {  
    
   String sCurrentLine;  
    
   br = new BufferedReader(new FileReader("C:testing.txt"));  
    
   while ((sCurrentLine = br.readLine()) != null){  
   System.*out*.println(sCurrentLine);  
   }  
   } catch (FileNotFoundException e){  
   e.printStackTrace();  
   } catch (IOException e) {  
   e.printStackTrace();  
   } finally {  
   try {  
   if (br != null)  
   br.close();  
   } catch (IOException ex){  
   ex.printStackTrace();  
   }  
   }  
   }  
  }
* Uncheck là loại expception xảy ra tại thời điểm thực thi chương trình

Thế nào là throw và throws trong java?

* Throw được sử dụng để ném ra một ngoại lệ cụ thể
* Throw thi được sử dụng để khai báo ngoại lệ
* Giữa các method liên quan đến nhau, checked thì được lan truyền, nghĩa là chỉ cần 1 trong số method đấy xử lý ngoại lệ là được còn unchecked thì không
* 1 phương thức được khai báo throws với class là checked thì trong phần thân phương thức mới được throw

Thế nào là try catch, try with resource trong java?

* try – catch là xử lý ngoại lệ trong java, try chứa những câu lệnh có thể xảy ra ngoại lệ, catch: nếu một ngoại lệ được ném ra từ khối try chương trình sẽ nhảy đến khối catch tương ứng với ngoại đó để xử lý ngoại lệ
* Trước java7 ta cần khai báo thêm finally để 1 đối tượng được đóng lại mỗi khi sử dụng nó, từ java7 trở đi ta không còn cần nó nữa
* Resource là 1 đối tượng cần phải được đóng lại mỗi khi sử dụng xong, ví dụ như ta mở 1 file hoặc kết nối với CSDL sau mỗi lần sử dụng ta phải đóng nó lại
* **public** **class** TryResourceJava7Example {
* **public** **static** **void** main(String[] args) **throws** IOException {
* **try** (BufferedReader br = **new** BufferedReader(**new** FileReader("D:\\gpcoder.txt"))) {
* String line;
* **while** ((line = br.readLine()) != **null**) {
* System.out.println(line);
* }
* } **catch** (IOException e) {
* e.printStackTrace();
* }
* }
* }

Xử lý ngoại lệ trong trường hợp ghi đè phương thức?

* Trường hợp 1: Phương thức lớp cha không khai báo exception thì ghi lớp con ghi đè nó chỉ được khai báo exception của unchecked còn checked thì không
* Trường hợp 2: Phương thức lớp cha ném ra exception thì khi phương thức của lớp con ghi đè nó phải được ném ra ngoại lệ cùng cấp hoặc dưới so với cái exception mà lớp cha ném ra

Làm cách nào để tạo ra 1 custom exception trong java?

* Custom exception là exception do người dùng tự định nghĩa, để xử lý những bài toán riêng của họ

Vì sao phải handle exception ?

* Bắt lỗi và đưa ra thông báo xử lý cho người dùng

Sub Query trong SQL

* Là câu truy vấn được nhúng trong 1 câu truy vấn khác, nó nằm trong dấu () và được truy vấn trước câu query chứa nó
* Trả lại 1 row 1 column, hay sử dụng cho toán tử : =, <>, <, >, <=, >=
* Trả lại nhiều row, 1 column: sử dụng cho IN và NOT IN
* Trả lại nhiều row, nhiều column

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | filtered | Extra |
|  |  |  |  |  |  |  |  |  |  |  |

CTE trong SQL

* Tạo ra 1 bảng tạm thời và ta có thể lấy dữ liệu từ bảng đấy
* Giả sử có 1 bài toán như sau: đó là lấy ra danh sách khách hàng ( nhân viên cũng có thể vừa làm nhân viên vừa làm khách hàng)
* Ban đầu ta tạo ra 1 CTE là a ta lấy ra danh sách không phải là admin, tiếp theo ta lấy tạo ra 1 CTE là b, trong b này t lấy ra usersId chỉ là khách hàng bằng cách:

+ ban đầu lấy ra userId có role là nhân viên từ bảng a, xong ta lấy ra userid trong b tại userId Not In userId có roleId = 2 (tức là nhân viên). Bây giờ tại CTE là b sẽ chứa những userID chỉ là Khách hàng

# Cấu trúc dữ liệu: Kiểu mảng

* Cấu trúc dữ liệu: là cái cách ta quản lý được dữ liệu và sử dụng chúng 1 cách hiệu quả nhất
* Cấu trúc dữ liệu được chia làm 2 loại là: tuyến tính và không tuyến tính

+ Tuyến tính là các phần tử dữ liệu được sắp xếp tuần tự hoặc tuyến tính và mỗi phần tử được liên kết với các phần tử kế tiếp hoặc sau nó

+ Phi tuyến tính: có cấu trúc phân cấp không rõ ràng, nghĩa là 1 phần tử có thể có phần tử con và cha

* Array: là 1 tập hợp các giá trị có cùng kiểu dữ liệu và mỗi 1 giá trị trong mảng sẽ được đánh chỉ số index để quản lý có thể truy cập bất kỳ giá trị nào của mảng
* Ưu điểm:

+ ta có thể truy cập 1 phần tử bất kỳ của mảng bằng chỉ số index và truy cập 1 cách nhanh chóng với độ phức tạp là O(1)

+ có thể lưu 1 tập hợp các giá trị có cùng kiểu dữ liệu

* Nhược điểm:

+ Kích thước mảng là cố định nên hạn chế sử dụng với tập dữ liệu thay đổi liên tục về kích thước

+ 1 số ngôn ngữ không có cơ chế tự dọn rác nên có thể gây rò rỉ bộ nhớ

* Kiểu mảng có thể tìm kiếm và sắp xếp được dựa trên những thuật toán mà ta áp dụng

# 1. Collection -Hệ thống phân cấp

**Mô tả hệ thống phân cấp Collection Framework trong java**

Phần 1: java.util.Collection Chứa danh sách các **đối tượng**

Cấp cao nhất là interface: Iterable

Cấp tiếp theo kế thừa Iterable là Collection, interface này có 3 interface chính kế thừa nó gồm:

* List: có 3 class kế thừa nó là Arraylist, vector và LinkedList
* Queue:

+ Có 1 interface kế thừa là deque và có class ArrayDeque và LinkedList impl nó

+ class PriorityQueue impl Queue

* Set: có 3 class kế thừa nó là HashSet, LinkedHashSet và TreeSet

Phần 2: java.util.Map chứa các cặp **key/value**

Cấp cao nhất là interface Map

Sẽ có 4 class impl từ nó, đó là: HashMap, LinkedHashMap, HashTable, TreeMap

**Thế nào là Iterable trong Java?**

Iterable là 1 interface trong java và nó chứa 1 phương thức iterator() trả về 1 đối tượng Iterator dùng để lặp qua các phần tử trong Collection

**Mô tả đặc điểm của 1 số interface collection:**

* List(danh sách):

+ tập hợp các phần tử có thứ tự và có thể trùng lặp giá trị

+ ta có thể đọc, thêm, sửa, xóa phần tử tại bất cứ vị trí nào

* Queue(hàng đợi):

+ là 1 tập hợp tuần tự nhưng ta chỉ có thể chạm vào phần tử đầu tiên của nó

+ Hoạt động theo nguyên tắc FIFO, nghĩa là 1 phần tử khi được chèn vào sẽ được xếp sau, và khi xóa sẽ xóa phần tử đầu tiên  
+ được phép chứa các phần tử trùng lặp và không cho phép phần tử null

* Deque(hàng đợi 2 đầu): nó cũng tương tự như queue nhưng nó có thể thao tác thêm vào, lấy ra và xóa phần tử cả 2 đầu của hàng đợi
* Set: là 1 tập hợp không tuần tự và không cho phép chứa các giá trị trùng lặp. Do không tuần tự nên ta không thể thao tác với phần tử thứ N, chỉ có thể thao tác với giá trị của tập hợp set đó nếu nó tồn tại
* SortedSet: nó là 1 interface con của set, nó cũng giống như set nhưng các phần tử trong SortedSet được sắp xếp theo thứ tự tăng dần
* Map: lưu trữ các cặp key/value, không được phép trùng lặp key, và mỗi key có thể có nhiều value

# 3. ArrayList

- ArrayList là 1 class trong collection và được impl List và extend từ AbstractList,

- nó lưu trữ các phần tử có cùng kiểu dữ liệu, cho phép trùng nhau, có thứ tự và nó là 1 mảng động

- thao tác truy cập đến các phần tử nhanh chóng nhờ vào việc lưu dữ liệu theo chỉ mục

- thêm và xóa sẽ lâu vì nó phải thiết lập lại chỉ mục index

- Ưu điểm:

+ Truy cập phần tử ngẫu nhiên nhanh  
+ Tự động tăng kích thước

+ Thêm, xóa vào cuối danh sách nhanh chóng  
- Nhược điểm:

+ Thao tác chèn, xóa phần tử ở giữa danh sách lâu do phải cấu trúc lại index

+ Cấp phát lại bộ nhớ liên tục khi thêm phần tử

# 4.LinkedList

* Cấu trúc dữ liệu kiểu danh sách liên kết là dạng cấu trúc dữ liệu gồm danh sách các node tạo thành 1 chuỗi, các node này chứa dữ liệu của node đó và địa chỉ tham chiếu đến node tiếp theo
* Kiểu danh sách liên kết đơn: nghĩa là các node sẽ chỉ chứa địa chỉ node tiếp theo của nó
* Kiểu danh sách liên kết đôi: nghĩa là các node sẽ chứa địa chỉ node đứng đằng trước và đứng đằng sau nó
* Ưu điểm: phù hợp với thao tác insert và remove vì nó không phải tổ chức lại như kiểu mảng
* Nhược điểm: khả năng truy cập phần tử bất kỳ chậm do nó phải tìm từng node
* LinkedList trong java được triển khai dựa trên kiểu danh sách liên kết đôi, các phần tử sẽ giống như các node chứa data và địa chỉ node đứng đằng trước và đằng sau nó
* Được phép chứa các phần tử trùng lặp, duy trì thứ tự khi được thêm vào
* Các phần tử trong Linked không nằm liên tiếp mà có thể nẳm bất kỳ trong ô nhớ, chính vì vậy khi thêm hoặc xóa thì các phần tử chỉ cần cập nhật lại địa chỉ tham chiếu node trước và sau của nó
* Giống nhau: đều chứa tập hợp các phần tử giống nhau về kiểu dữ liệu và duy trì thứ tự khi được thêm vào. Đều được impl từ List
* Khác nhau giữa LinkedList và ArrayList

|  |  |  |
| --- | --- | --- |
|  | ArrayList | LinkedList |
| Cấu trúc dữ liệu | Sử dụng cấu trúc dữ liệu dạng mảng động, các phần tử đều được liên kết với 1 chỉ mục và vị trí của chúng là liên tiếp nhau trong 1 vùng nhớ | Sử dụng cấu trúc dữ liệu dưới dạng danh sách liên kết, các phần tử là tập hợp các node tạo thành chuỗi và tham chiếu lẫn nhau. Mỗi node sẽ chứa giá trị, địa chỉ node trước và sau của nó |
| Thao tác thêm và xóa | Mỗi khi thêm hoặc xóa, mảng phải sắp xếp lại index nên sẽ chậm O(n) | LinkedList sẽ nhanh hơn bởi vì nó không cần sắp xếp lại các phần tử mà chỉ cần cập nhật lại node tham chiếu trước và sau nó. O(1) |
|  | ArrayList có thể truy xuất ngẫu nhiên | LinkedList thì không, nó phải đi từ đầu đến cuối để tìm phần tử |
|  | ArrayList sử dụng ít bộ nhớ hơn | LinkedList sử dụng nhiều hơn bởi mỗi 1 phần tử nó lưu trữ nhiều dữ liệu hơn ArrayList |
|  | Phù hợp cho việc lưu trữ dữ liệu và truy xuất | Phù hợp cho việc thêm và xóa dữ liệu |

# 5.Set

* Set là 1 tập hợp các phần tử chứa những giá trị không trùng nhau và không theo thứ tự
* HashSet được khởi tạo từ HashMap, nó cũng được đưa vào bucker, chỉ khác 1 điều là value của nó luôn là null

# 6.Map

* Map dùng để lưu trữ và truy xuất dữ liệu theo cặp key/value
* Map chỉ chứa key duy nhất
* Nguyên lý hoạt động khi thêm vào của HashMap: khi ta thêm 1 phần tử vào HashMap nó sẽ hashing(băm) key ra giá trị băm hashcode(là 1 số nguyên dương). Sau đó sẽ tính chỉ số index (của thùng băm) dựa trên mã băm này(giá trị băm % 16) và đưa Entry này vào bucket. Bucket là nơi chứa những Entry có cùng mã băm, mỗi 1 node trong bucket sẽ chứa giá trị băm, key, value, và địa chỉ của node tiếp theo trong bucket đó
* Khi tìm kiếm key trong HashMap, thì nó sẽ hashing key đó ra và tính chỉ số index. Sau đó đi tới bảng băm với index đã được tính vào bắt đầu so sánh các node trong bucket, giả sử node đầu tiên không tìm thấy nó sẽ đi đến node tiếp theo …. Vì các node trong bucker này là 1 danh sách liên kết đơn

# 7. queue

* Là tập hợp các phần tử có thứ tự, hoạt động theo nguyên lý FIFO, bạn chỉ có thể gọi và xóa phần tử đứng đầu trong nó. Chó phép phần tử trùng lặp và không được phép giá trị null

# Design pattern

# Kinh nghiệm code => chuyên nghiệp + tái sử dụng cao ( code rõ ràng, sáng sủa, dễ đọc )

# Tổng có 23 mẫu, tương ứng với 23 cách lập trình

# Được chia làm 3 loại:

# + hàm tạo: 5 mẫu

# + methods: 7 mẫu

# + behaviors (hành vi code): 11 mẫu

# Cách học

# + Trong trường hợp nào thì chúng ta sử dụng tới mẫu đó

# + Cách code như thế nào

# Creational – Singleton

# Tạo ra 1 class object khi chạy dự án và chỉ tạo ra 1 object duy nhất (static)

# Tạo ra 1 class và ta chỉ muốn tồn tại 1 và chỉ 1 instance từ cái class đó. Và instance này như 1 biến toàn cục, có thể sử dụng bất cứ đâu trong chương trình

# Phát triển như thế nào????

# Tìm hiểu trong TH nào thì tạo ra Singleton design pattern

# Ví dụ: Xây dựng 1 dự án quản lý bán hàng

# + Tạo ra class Danh mục sản phẩm: id, catName => sẽ phải tạo ra nhiều đối tượng chúng ta không được gọi là Singleton

# + Sản phẩm: id, title, price, idCat

# + Tạo ra 1 đối tượng DataMgr => chỉ cần tạo 1 object từ class DataMgr = > singleton object (Singleton design pattern )

# Quản lý danh mục sản phẩm

# Quản lý sản phẩm

# + Class Main

# Singleton là kiểu thiết kế mà nó đảm bảo chỉ có 1 instance được tạo ra và nó cung cấp 1 method duy nhất để có thể truy suất đến instance đó

# Ứng dụng thực tế:

# Ghi lại lịch sử đăng nhập

# Sử dụng lại làm file config

# 3. Creational - Factory Method Pattern

# Creational -Dependency Injection

# Dependency Inversion là 1 quy tắc lập trình mà trong đó các module cấp cao không nên phụ thuộc và module cấp thấp. 2 module này nên phụ thuộc và abstraction. Còn thêm quy định là các class giao tiếp với nhau thôn qua inteface chứ không được impl

# Ioc là hướng đi, DIP là định hình cụ thể của hướng đi, DI là 1 thực hiện cụ thể

# Dependency injection là 1 design pattern, một kỹ thuật cho phép xóa bỏ sự phụ thuộc giữa các module, làm cho ứng dụng dễ dàng thay đổi hơn trong việc thay đổi module, bảo trì code và testing.