Cần ôn:

1. Kỹ thuật lập trình ở năm 1 (C++ và Java)

Là lại các bài lab môn kỹ thuật lập trình

1. Datatructure , DataBase System ,OOP ở năm 2
2. Software Testing ở năm 4

Read book : SCJP SUN CERTIFIED PROGRAMMER FOR JAVA

câu hỏi phỏng vấn:

1. Nói về các từ khóa trong Java
2. **Final key word for object?** : 'final' simply makes the object *reference* unchangeable. The object it points to is not immutable by doing this. INSTANCE can never refer to another object, but the object it refers to may change state.
3. Thread stop rồi có start lại được ko ?

Không.

1. Compare Integer in Java: use compareTo or equal method

Integer x1 = 23;

Integer y1 = 23;

Integer x2 = 127;

Integer y2 = 127;

Integer x3 = 128;

Integer y3 = 128;

System.***out***.println(x1 == y1);//result is true

System.***out***.println(x2 == y2);// result is true

System.***out***.println(x3 == y3);// result is false

Reason: java 7 doc If the value p being boxed is true, false, a byte, or a char in the range \u0000 to \u007f, or an int or short number between -128 and 127 (inclusive), then let r1 and r2 be the results of any two boxing conversions of p. It is always the case that r1 == r2.

1. class con có thể nhìn thấy biến private của class cha không?

Không.

1. Access Modifiers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifier** | **Class** | **Package** | **Subclass** | **World** |
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| *no modifier* | Y | Y | N | N |
| private | Y | N | N | N |

1. **Overload and Overwrite**:

Overload: The Java programming language supports overloading methods, and Java can distinguish between methods with different method signatures. This means that methods within a class can have the same name if they have different parameter lists (there are some qualifications to this that will be discussed in the lesson titled "Interfaces and Inheritance").

the compiler does not consider return type when differentiating methods, so you cannot declare two methods with the same signature even if they have a different return type.

Overwrite : An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method.

1. In addition, an interface can contain constant declarations. All constant values defined in an interface are implicitly public, static, and final. Once again, you can omit these modifiers.
2. **Interface have same method signature** ( read java doc)

It is permitted for a single method declaration in a class to implement methods of more than one superinterface. For example, in the code:

interface Fish { int getNumberOfScales(); }

interface Piano { int getNumberOfScales(); }

class Tuna implements Fish, Piano {

// You can tune a piano, but can you tuna fish?

int getNumberOfScales() { return 91; }

}

the method getNumberOfScales in class Tuna has a name, signature, and return type that matches the method declared in interface Fish and also matches the method declared in interface Piano; it is considered to implement both.

On the other hand, in a situation such as this:

interface Fish { int getNumberOfScales(); }

interface StringBass { double getNumberOfScales(); }

class Bass implements Fish, StringBass {

// This declaration cannot be correct, no matter what type is used.

public ??? getNumberOfScales() { return 91; }

}

It is impossible to declare a method named getNumberOfScales whose signature and return type are compatible with those of both the methods declared in interface Fish and in interface StringBass, because a class cannot have multiple methods with the same signature and different primitive return types . Therefore, it is impossible for a single class to implement both interface Fish and interface StringBass .

1. The interface body can contain abstract methods, default methods, and static methods. An abstract method within an interface is followed by a semicolon, but no braces (an abstract method does not contain an implementation). Default methods are defined with the default modifier, and static methods with the static keyword. All abstract, default, and static methods in an interface are implicitly public, so you can omit the public modifier.
2. **Is Java “pass-by-reference” or “pass-by-value”?**

Java is always pass-by-value. The difficult thing to understand is that Java passes objects as references and those references are passed by value.

public static void main( String[] args ){

Dog aDog = new Dog("Max");

foo(aDog);

if( aDog.getName().equals("Max") ){ //true

System.out.println( "Java passes by value." );

}else if( aDog.getName().equals("Fifi") ){

System.out.println( "Java passes by reference." );

}

}

public static void foo(Dog d) {

d.getName().equals("Max"); // true

d = new Dog("Fifi");

d.getName().equals("Fifi"); // true

}

In this example aDog.getName() will still return "Max". The value aDog within main is not overwritten in the function foo with the Dog "Fifi" as the object reference is passed by value. If it were passed by reference, then the aDog.getName() in main would return "Fifi" after the call to foo.

Likewise:

Dog aDog = new Dog("Max");

foo(aDog);

aDog.getName().equals("Fifi"); // true

public void foo(Dog d) {

d.getName().equals("Max"); // true

d.setName("Fifi");

}

1. **static block** ?

Static Initializers: It's a static initializer. It's executed when the class is loaded.

1. Is Java Multiple Inheritance or single inheritance ?

Single

1. class method and instance method

Static methods, which have the static modifier in their declarations, should be invoked with the class name, without the need for creating an instance of the class.

Instance methods can access instance variables and instance methods directly.

Instance methods can access class variables and class methods directly.

Class methods can access class variables and class methods directly.

Class methods cannot access instance variables or instance methods directly—they must use an object reference. Also, class methods cannot use the this keyword as there is no instance for this to refer to.

1. **abstract class:**

An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

1. **abstract method:** is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

abstract void moveTo(double deltaX, double deltaY);

If a class includes abstract methods, then the class itself must be declared abstract, as in:

public abstract class GraphicObject {

// declare fields

// declare nonabstract methods

abstract void draw();

}

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

1. Java Collection



A collection that contains no duplicate elements. More formally, sets contain no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. As implied by its name, this interface models the mathematical set abstraction.

Set \_\_ A collection that contains no duplicate elements. More formally, sets contain no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. As implied by its name, this interface models the mathematical set abstraction..

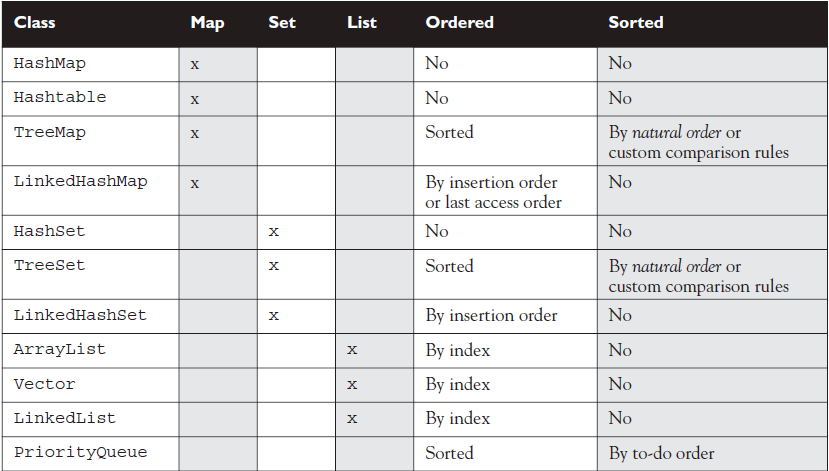
List — An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Unlike sets, lists typically allow duplicate elements. More formally, lists typically allow pairs of elements e1 and e2 such that e1.equals(e2), and they typically allow multiple null elements if they allow null elements at all. It is not inconceivable that someone might wish to implement a list that prohibits duplicates, by throwing runtime exceptions when the user attempts to insert them, but we expect this usage to be rare.

Queue — a collection used to hold multiple elements prior to processing. Besides basic Collection operations, a Queue provides additional insertion, extraction, and inspection operations.

Queues typically, but do not necessarily, order elements in a FIFO (first-in, first-out) manner. Among the exceptions are priority queues, which order elements according to a supplied comparator or the elements' natural ordering. Whatever the ordering used, the head of the queue is the element that would be removed by a call to remove or poll. In a FIFO queue, all new elements are inserted at the tail of the queue. Other kinds of queues may use different placement rules. Every Queue implementation must specify its ordering properties. Also see The Queue Interface section.

Deque — a collection used to hold multiple elements prior to processing. Besides basic Collection operations, a Deque provides additional insertion, extraction, and inspection operations.



1. Difference between ArrayList and Vector:

A Vector is basically the same as an ArrayList, but Vector methods are synchronized

for thread safety.

1. Different between ArrayList and LinkedList:

A LinkedList is ordered by index position, like ArrayList, except that the elements are doubly-linked to one another.

1. Different between HashSet and LinkedHashSet:

A LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements. Use this class instead of HashSet when you care about the iteration order.

1. What is importance notice when using HashSet or LinkedHashSet

the objects you add to them must override hashCode(). If they don’t override hashCode(), the default Object. hashCode() method will allow multiple objects that you might consider "meaningfully equal" to be added to your "no duplicates allowed" set

1. What is the difference between Set and List ?

List<E>:

An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Set<E>:

A collection that contains no duplicate elements. More formally, sets contain no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. As implied by its name, this interface models the mathematical set abstraction.

1. Different between HashMap and HashTable:

Hashmap: permits null values and the null key. HashMap not synchronized.

HashTable: Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method. Hashtable is synchronized.

╔══════════════╦═════════════════════╦═══════════════════╦══════════════════════╗

║ Property ║ HashMap ║ TreeMap ║ LinkedHashMap ║

╠══════════════╬═════════════════════╬═══════════════════╬══════════════════════╣

║ ║ no guarantee order ║ sorted according ║ ║

║ Order ║ will remain constant║ to the natural ║ insertion-order ║

║ ║ over time ║ ordering ║ ║

╠══════════════╬═════════════════════╬═══════════════════╬══════════════════════╣

║ Get/put ║ ║ ║ ║

║ remove ║ O(1) ║ O(log(n)) ║ O(1) ║

║ containsKey ║ ║ ║ ║

╠══════════════╬═════════════════════╬═══════════════════╬══════════════════════╣

║ ║ ║ NavigableMap ║ ║

║ Interfaces ║ Map ║ Map ║ Map ║

║ ║ ║ SortedMap ║ ║

╠══════════════╬═════════════════════╬═══════════════════╬══════════════════════╣

║ ║ ║ ║ ║

║ Null ║ allowed ║ only values ║ allowed ║

║ values/keys ║ ║ ║ ║

╠══════════════╬═════════════════════╩═══════════════════╩══════════════════════╣

║ ║ Fail-fast behavior of an iterator cannot be guaranteed ║

║ Fail-fast ║ impossible to make any hard guarantees in the presence of ║

║ behavior ║ unsynchronized concurrent modification ║

╠══════════════╬═════════════════════╦═══════════════════╦══════════════════════╣

║ ║ ║ ║ ║

║Implementation║ buckets ║ Red-Black Tree ║ double-linked ║

║ ║ ║ ║ buckets ║

╠══════════════╬═════════════════════╩═══════════════════╩══════════════════════╣

║ Is ║ ║

║ synchronized ║ implementation is not synchronized ║

╚══════════════╩════════════════════════════════════════════════════════════════╝

1. What is signature?

Two of the components of a method declaration comprise the method signature—the method's name and the parameter types.

1. Does a method's signature in Java include its return type?

Definition: Two of the components of a method declaration comprise the method signature—the method's name and the parameter types.

1. Can we override static method?

No, you can't override the static method because they are the part of class not object.

1. What does variable define in an interface must be?

public, static, and final

1. Can a private method be overridden by a subclass?

That's an interesting question, but the answer is technically no. Since the subclass, as we've seen, cannot inherit a private method, it therefore cannot override the method.

1. Can interface method be static ?

Interface methods must not be static.

1. Is the subclass can access private field of super class ?

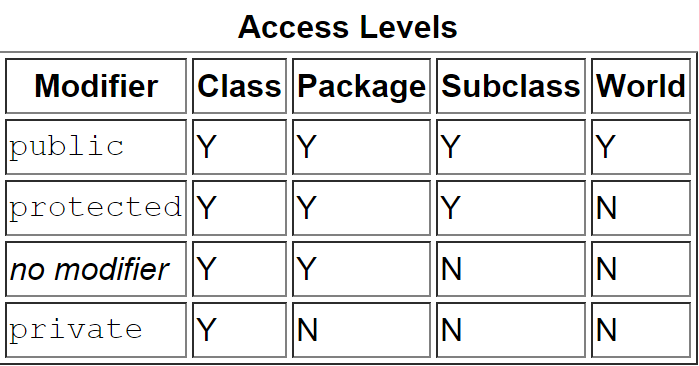
No.

1. Is the subclass can access protected field of super class ?

Yes

1. What is default modifier in Java?

If a class has no modifier (the default, also known as package-private), it is visible only within its own package.



1. What is synchronized keyword mean ?

synchronized keyword indicates that a method can be accessed by only one thread at a time.

1. What is Variable Argument Lists (var-args) ?

As of 5.0, Java allows you to create methods that can take a variable number of

Arguments. Example:

void doStuff(int... x) { } // expects from 0 to many ints

// as parameters

void doStuff2(char c, int... x) { } // expects first a char,

// then 0 to many ints

void doStuff3(Animal... animal) { } // 0 to many Animals

1. What is reference type in java?

A reference variable is used to refer to (or access) an object. A reference variable is declared to be of a specific type and that type can never be changed. A reference variable can be used to refer to any object of the declared type, or of a subtype of the declared type (a compatible type).

1. What is Instance Variables ?

Fields that have the static modifier in their declaration are called static fields or class variables.

1. What is Variable shadowing ?

In computer programming, variable shadowing occurs when a variable declared within a certain scope (decision block, method, or inner class) has the same name as a variable declared in an outer scope.

1. What is transient variables ?

If you mark an instance variable as transient, you're telling the JVM to skip (ignore) this variable when you attempt to serialize the object containing it.

1. Serialization is one of the coolest features of Java; it lets you save (sometimes called

"flatten") an object by writing its state (in other words, the value of its instance

variables) to a special type of I/O stream.

1. Is Polymorphic method invocations apply to class method ?

No. Polymorphic method invocations apply only to instance methods.

1. Tính chất của lập trình hướng đối tượng

* Tính trừu tượng (abstraction): Đây là khả năng của chương trình bỏ qua hay không chú ý đến một số khía cạnh của thông tin mà nó đang trực tiếp làm việc lên, nghĩa là nó có khả năng tập trung vào những cốt lõi cần thiết. Mỗi đối tượng phục vụ như là một "động tử" có thể hoàn tất các công việc một cách nội bộ, báo cáo, thay đổi trạng thái của nó và liên lạc với các đối tượng khác mà không cần cho biết làm cách nào đối tượng tiến hành được các thao tác. Tính chất này thường được gọi là sự trừu tượng của dữ liệu.Tính trừu tượng còn thể hiện qua việc một đối tượng ban đầu có thể có một số đặc điểm chung cho nhiều đối tượng khác như là sự mở rộng của nó nhưng bản thân đối tượng ban đầu này có thể không có các biện pháp thi hành. Tính trừu tượng này thường được xác định trong khái niệm gọi là lớp trừu tượng hay lớp cơ sở trừu tượng.
* Tính đóng gói (encapsulation) và che giấu thông tin (information hiding): Tính chất này không cho phép người sử dụng các đối tượng thay đổi trạng thái nội tại của một đối tượng. Chỉ có các phương thức nội tại của đối tượng cho phép thay đổi trạng thái của nó. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã. Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.
* Tính đa hình (polymorphism): Thể hiện thông qua việc gửi các thông điệp (message). Việc gửi các thông điệp này có thể so sánh như việc gọi các hàm bên trong của một đối tượng. Các phương thức dùng trả lời cho một thông điệp sẽ tùy theo đối tượng mà thông điệp đó được gửi tới sẽ có phản ứng khác nhau. Người lập trình có thể định nghĩa một đặc tính (chẳng hạn thông qua tên của các phương thức) cho một loạt các đối tượng gần nhau nhưng khi thi hành thì dùng cùng một tên gọi mà sự thi hành của mỗi đối tượng sẽ tự động xảy ra tương ứng theo đặc tính của từng đối tượng mà không bị nhầm lẫn.Thí dụ khi định nghĩa hai đối tượng "hinh\_vuong" và "hinh\_tron" thì có một phương thức chung là "chu\_vi". Khi gọi phương thức này thì nếu đối tượng là "hinh\_vuong" nó sẽ tính theo công thức khác với khi đối tượng là "hinh\_tron".
* Tính kế thừa (inheritance): Đặc tính này cho phép một đối tượng có thể có sẵn các đặc tính mà đối tượng khác đã có thông qua kế thừa. Điều này cho phép các đối tượng chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại. Tuy nhiên, không phải ngôn ngữ định hướng đối tượng nào cũng có tính chất này.

1. Đa hình là gì

Là khả năng các đối tượng khác nhau sẽ có những cách trả lời khác nhau cho cùng một thông điệp.

Hay

Cùng 1 lệnh gởi thông điệp đến đối tượng thông qua cùng 1 tham khảo nhưng ở vị trí/thời điểm khác nhau có thể kích hoạt việc thực thi tác vụ khác nhau của các đối tượng khác nhau.

1. Cho đoạn code sau:

public class TestAnimals {

public static void main (String [] args) {

Animal a = new Animal();

Animal b = new Horse(); //Animal ref, but a Horse object

a.eat(); // Runs the Animal version of eat()

b.eat(); // Runs the Horse version of eat()

}

}

class Animal {

public void eat() {

System.out.println("Generic Animal Eating Generically");

}

}

class Horse extends Animal {

public void eat() {

System.out.println("Horse eating hay, oats, "+ "and horse treats");

}

public void buck() { }

}

Đặt đoạn code sau trong hàm main

Animal c = new Horse();

c.buck(); // Can't invoke buck();

lúc build có lỗi hay ko?

Answer: Có. Remember, the compiler will allow only methods in class Animal

to be invoked when using a reference to an Animal.

1. In overwrite method, the return type could be the same as, or a subtype of, the return type declared in the original overridden method in the superclass, isn’t it?

Yes.

1. In overwrite method , The access level CAN be less restrictive than that of the overridden method, isn’t it ?

Yes.

1. A method marked final can be overrided ?

No.

1. A method marked static can be overrided ?

No

1. How to Invoking a Superclass Version of an Overridden Method ?

Use super key word

Example: super.printYourself();

1. This code run Ok or Not

class Animal {

public void eat() throws Exception {

// throws an Exception

}

}

class Dog2 extends Animal {

public void eat() { /\* no Exceptions \*/}

public static void main(String [] args) {

Animal a = new Dog2();

Dog2 d = new Dog2();

d.eat(); // ok

a.eat();

}

}

1. The overriding method must NOT throw checked exceptions that are new or broader than those declared by the overridden method. For example, a method that declares a FileNotFoundException cannot be overridden by a method that declares a SQLException, Exception, or any other non-runtime exception unless it's a subclass of FileNotFoundException.

True or False

true

1. The overriding method CAN throw any unchecked (runtime) exception, regardless of whether the overridden method declares the exception. True or False ?

True.

1. A method can be overloaded in the same class or in a subclass. True or False

True

1. In override, The return type of overriding in subclass could be different the original overridden method in the superclass. True of False?

False. The return type must be the same as, or a subtype of, the return type declared

in the original overridden method in the superclass.

1. Give the code:

public class Foo {

public void doStuff(int y, String s) { }

public void moreThings(int x) { }

}

class Bar extends Foo {

public void doStuff(int y, String s) throws IOException { }

}

Does this code ok?

Answer: This code make error. Because this is override. And method in subclass cannot throw new Exception

1. Give the code:

public class Foo {

public void doStuff(int y, String s) { }

public void moreThings(int x) { }

}

class Bar extends Foo {

public void doStuff(int y, long s) throws IOException { }

}

Does this code ok?

Answer: It's tempting to see the IOException as the problem, because the

overridden doStuff() method doesn’t declare an exception, and IOException is checked

by the compiler. But the doStuff() method is not overridden! Subclass Bar overloads the

doStuff() method, by varying the argument list, so the IOException is fine.

1. Give the code:

public class Foo {

public void doStuff(int y, int s) { }

}

class Bar extends Foo {

public void doStuff(Integer y, Integer s) { }

}

Is this overload ?

Yes

1. Give the code

public class Animal {

public void eat(int i) {

System.out.println("Animal");

}

}

public class Dog extends Animal {

public void eat(Integer i) {

System.out.println("Dog");

};

public static void main(String[] args) {

Dog d = new Dog();

d.eat(1);// Animal

d.eat(new Integer(1));// Dog

}

}

What is the result of above code ?

Answer:

Animal

Dog

1. Give the code:

public class Animal {

public void eat(int i) {

System.out.println("Animal");

}

}

public class Dog extends Animal {

// public void eat(Integer i) {

// System.out.println("Dog");

// };

public static void main(String[] args) {

Dog d = new Dog();

d.eat(1);// Animal

d.eat(new Integer(1));// Dog

}

}

What is the result of above code ?

Answer:

Animal

Animal

1. Give the code

class Animal {

}

class Horse extends Animal {

}

public class UseAnimals {

public void doStuff(Animal a) {

System.out.println("In the Animal version");

}

public void doStuff(Horse h) {

System.out.println("In the Horse version");

}

public static void main(String[] args) {

UseAnimals ua = new UseAnimals();

Animal animalObj = new Animal();

Horse horseObj = new Horse();

Animal animalRefToHorse = new Horse();

ua.doStuff(animalObj);

ua.doStuff(horseObj);

ua.doStuff(animalRefToHorse);

}

}

What is the result of this code ?

Answer:

In the Animal version

In the Horse version

In the Animal version

1. Give the code

public class Animal {

public void eat() {

System.out.println("Generic Animal Eating Generically");

}

}

public class Horse extends Animal {

public void eat() {

System.out.println("Horse eating hay ");

}

public void eat(String s) {

System.out.println("Horse eating " + s);

}

}

And 6 case:

a) Animal a = new Animal();

a.eat();

b) Horse h = new Horse();

h.eat();

c) Animal ah = new Horse();

ah.eat();

d) Horse he = new Horse();

he.eat("Apples");

e) Animal a2 = new Animal();

a2.eat("treats");

f) Animal ah2 = new Horse();

ah2.eat("Carrots");

Result:

a) Generic Animal Eating Generically

b) Horse eating hay

c) Horse eating hay

d) Horse eating Apples

e) Compiler error!

f) Compiler error!

1. What are checked exceptions ?

Represent invalid conditions in areas outside the immediate control of the program (invalid user input, database problems, network outages, absent files).

Checked exceptions are subclasses of Exception.

1. What are unchecked exception?

Unchecked runtime exceptions represent conditions that, generally speaking, reflect errors in your program's logic and cannot be reasonably recovered from at run time.

Unchecked exception are subclasses of RuntimeException, and are usually implemented using IllegalArgumentException, NullPointerException, or IllegalStateException

1. Different between checked exception and unchecked exception

Điểm khác biệt giữa các lớp checked và unchecked expcetion chính là thời điểm xác định được expcetion có thể xảy ra. Đối với checked exception, việc kiểm tra được thực hiện ngay thời điểm compile time, một số IDE sẽ giúp chúng ta bằng cách hiển thị lỗi cú pháp nếu ta gọi một method throw ra bất kỳ checked exception nào mà không được catch.

1. Can we override a method marked final ?

No.

1. Can we override a method marked static ?

No.

1. If a method can't be inherited, you cannot override it.

True.

1. How to invoke a Superclass Version of an Overridden Method ?

Use super keyword.

1. Give the code

public class Animal {

public void eat() { };

}

public class Dog2 extends Animal {

public void eat() throws Exception{ /\* do something\*/}

public static void main(String [] args) {

Animal a = new Dog2();

Dog2 d = new Dog2();

try {

d.eat();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

} // ok

a.eat();

}

}

Does this code work ok?

Answer: No. We get error, Exception Exception is not compatible with throws clause in Animal.eat()

1. Give the code

public class Animal {

public void eat() throws Exception{ };

}

public class Dog2 extends Animal {

public void eat(){ /\* no Exceptions \*/}

public static void main(String [] args) {

Animal a = new Dog2();

Dog2 d = new Dog2();

try {

d.eat();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

} // ok

a.eat();

}

}

Does this code work ok ?

Answer: No. Missing throw declaration or try/catch block

1. Does this code work ok

class Animal {

public void eat(){ /\* no Exceptions \*/}

}

public class Dog2 extends Animal {

public String eat(){ /\* no Exceptions \*/}

}

Answer: Yes. But this is not an override because of the return type, not an overload

either because there’s no change in the argument list.

1. Does this code work Ok?

public class Foo {

public void doStuff(int y, String s) { }

public void moreThings(int x) { }

}

class Bar extends Foo {

public void doStuff(int y, long s) throws IOException { }

}

Answer: Yes. It's tempting to see the IOException as the problem, because the

overridden doStuff() method doesn’t declare an exception, and IOException is checked

by the compiler. But the doStuff() method is not overridden! Subclass Bar overloads the

doStuff() method, by varying the argument list, so the IOException is fine.

1. What is the output of this code ?

class Animal { }

class Horse extends Animal { }

class UseAnimals {

public void doStuff(Animal a) {

System.out.println("In the Animal version");

}

public void doStuff(Horse h) {

System.out.println("In the Horse version");

}

public static void main (String [] args) {

UseAnimals ua = new UseAnimals();

Animal animalObj = new Animal();

Horse horseObj = new Horse();

ua.doStuff(animalObj);

ua.doStuff(horseObj);

}

}

Answer:

in the Animal version

in the Horse version

1. What is the output of this code:

class Animal { }

class Horse extends Animal { }

class UseAnimals {

public void doStuff(Animal a) {

System.out.println("In the Animal version");

}

public void doStuff(Horse h) {

System.out.println("In the Horse version");

}

public static void main (String [] args) {

UseAnimals ua = new UseAnimals();

Animal animalObj = new Animal();

Horse horseObj = new Horse();

Animal animalRefToHorse = new Horse();

ua.doStuff(animalRefToHorse); }

}

Answer: in the Animal version.

1. Is a method both overloaded and overridden ?

public class Animal {

public void eat() {

System.out.println("Generic Animal Eating Generically");

}

}

public class Horse extends Animal {

public void eat() {

System.out.println("Horse eating hay ");

}

public void eat(String s) {

System.out.println("Horse eating " + s);

}

}

1. Give two class as bellow:

public class Animal {

public void eat() {

System.out.println("Generic Animal Eating Generically");

}

}

public class Horse extends Animal {

public void eat() {

System.out.println("Horse eating hay ");

}

public void eat(String s) {

System.out.println("Horse eating " + s);

}

}

Give the result of code bellow:

a) Animal a = new Animal();

a.eat();

b) Horse h = new Horse();

h.eat();

c) Animal ah = new Horse();

ah.eat();

d) Horse he = new Horse();

he.eat("Apples");

e) Animal a2 = new Animal();

a2.eat("treats");

f) Animal ah2 = new Horse();

ah2.eat("Carrots");

Answer:

1. Generic Animal Eating Generically
2. Horse eating hay
3. Horse eating hay

Polymorphism works—the actual object type (Horse), not the

reference type (Animal), is used to determine which eat() is called.

1. Horse eating Apples

The overloaded eat(String s) method is invoked.

1. Compiler error! Compiler sees that Animal class doesn't have an

eat() method that takes a String.

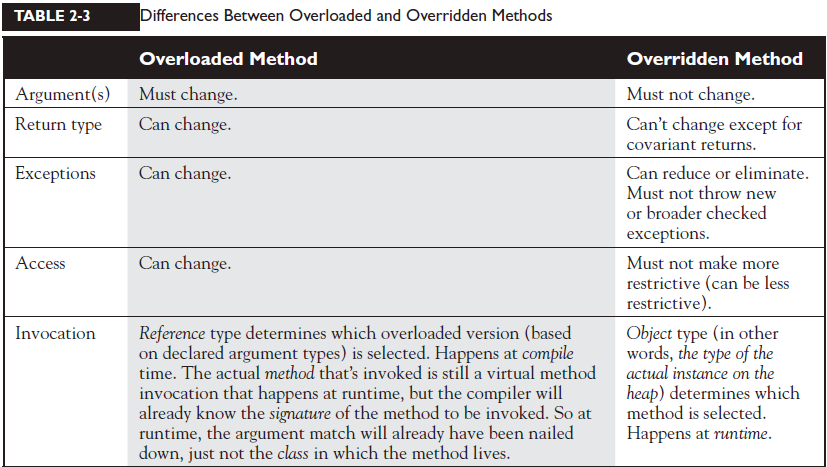
1. Compiler error! Compiler still looks only at the reference, and sees

that Animal doesn’t have an eat() method that takes a String.

Compiler doesn’t care that the actual object might be a Horse at

runtime.

1. Remember this table



1. Give the code

class Animal {

void makeNoise() {System.out.println("generic noise"); }

}

class Dog extends Animal {

void makeNoise() {System.out.println("bark"); }

void playDead() { System.out.println("roll over"); }

}

class CastTest2 {

public static void main(String [] args) {

Animal [] a = {new Animal(), new Dog(), new Animal() };

for(Animal animal : a) {

animal.makeNoise();

if(animal instanceof Dog) {

animal.playDead(); // try to do a Dog behavior ?

}

}

}

}

Is this code above work ok ?

Answer: No. The compiler will announce error cannot find symbol. We could modify this code :

if(animal instanceof Dog) {

Dog d = (Dog) animal; // casting the ref. var.

d.playDead();

}

1. What happen to this code ?

class Animal { }

class Dog extends Animal { }

class DogTest {

public static void main(String [] args) {

Animal animal = new Animal();

Dog d = (Dog) animal; // compiles but fails later

}

}

Answer: It can be maddening! This code compiles! When we try to run it, we'll get an

exception something like this:

java.lang.ClassCastException

1. Give the code

class Animal { }

class Dog extends Animal { }

class DogTest {

public static void main(String [] args) {

Animal animal = new Animal();

Dog d = (Dog) animal;

String s = (String) animal; // animal can't EVER be a String}

}

What happen with this code?

Answer:

The compiler announce error: inconvertible types.

1. May overloaded methods have different access modifiers?

Answer: yes. View above table.

1. May overloaded methods throw different exceptions ?

Answer:yes. View above.

1. What is covariant return type?

As of Java 5, you're allowed to change the return type in the overriding method as long as the new return type is a subtype of the declared return type of the overridden (superclass) method.

Example:

// Classes used as return types:

class A {

}

class B extends A {

}

// "Class B is more narrow than class A"

// Classes demonstrating method overriding:

class C {

A getFoo() {

return new A();

}

}

class D extends C {

//Overriding getFoo() in father class C

B getFoo() {

return new B();

}

}

1. Primitive Instance Variables. Give the code

public class BirthDate {

int year; // Instance variable

public static void main(String [] args) {

BirthDate bd = new BirthDate();

bd.showYear();

}

public void showYear() {

System.out.println("The year is " + year);

}

}

What is the result of code above?

Answer:0

1. Object Reference Instance Variables. Give the code

public class Book {

private String title; // instance reference variable

public String getTitle() {

return title;

}

public static void main(String [] args) {

Book b = new Book();

System.out.println("The title is " + b.getTitle());

}

}

What is the result of code above?

Answer: The title is null

1. Give the code

public class BirthDays {

static int [] year = new int[10];

public static void main(String [] args) {

for(int i=0;i<100;i++)

System.out.println("year[" + i + "] = " + year[i]);

}

}

Answer:

year[0] = 0

year[1] = 0

year[2] = 0

year[3] = 0

year[4] = 0

year[5] = 0

year[6] = 0

year[7] = 0

year[8] = 0

year[9] = 0

1. Give the code

public class BirthDate {

static int[] year ;

public static int[] getYear() {

return year;

}

public static void setYear(int[] year) {

BirthDate.year = year;

}

public static void main(String[] args) {

BirthDate birthDate = new BirthDate();

System.out.println(birthDate.getYear());

}

}

What is the result of code above ?

Answer: null

1. Local Primitives. Give the code

public class TimeTravel {

public static void main(String [] args) {

int year ;

System.out.println("The year is " + year);

}

}

What is the result of code above?

Answer: Compiler error.

1. Local Object References. Give the code:

import java.util.Date;

public class TimeTravel {

public static void main(String [] args) {

Date date;

if (date == null)

System.out.println("date is null");

}

}

What is the result of code above ?

Answer:

TimeTravel.java:5: Variable date may not have been initialized.

if (date == null)

1 error

1. Give the code

import java.awt.Dimension;

class ReferenceTest {

public static void main (String [] args) {

Dimension a = new Dimension(5,10);

System.out.println("a.height = " + a.height);

Dimension b = a;

b.height = 30;

System.out.println("a.height = " + a.height +" after change to b");

}

}

What is the result of code above ?

a.height = 10

a.height = 30 after change to b

1. What is difference of stateless session bean and stateful session bean ?
2. What happen if a class Implementing multiple interfaces having same method ?

Follow chapter 9.4.1.3. Inheriting Methods with Override-Equivalent Signatures in Java Language Specification

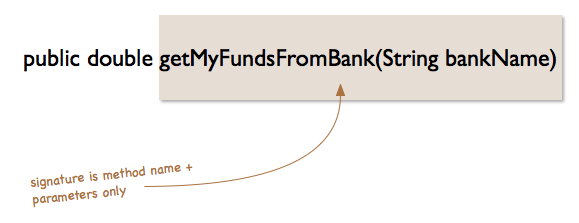
It is possible for an interface to inherit several methods with override-equivalent signatures (§8.4.2). Such a situation does not in itself cause a compile-time error. The interface is considered to inherit all the methods.

However, one of the inherited methods must be return-type-substitutable for every other inherited method; otherwise, a compile-time error occurs. (The throws clauses do not cause errors in this case.)

There might be several paths by which the same method declaration is inherited from an interface. This fact causes no difficulty and never, of itself, results in a compile-time error.

1. Does a method's signature in Java include its return type?

Answer:no.



1. What is difference between For-each vs Iterator in Java ?

In iterator the remove() is available to remove the content/item but in for loop remove() not available.

You might need to use iterators if you need to modify collection in your loop. First approach will throw exception.

for (String i : list) {

System.out.println(i);

list.remove(i); // throws exception

}

Iterator it=list.iterator();

while (it.hasNext()){

System.out.println(it.next());

it.remove(); // valid here

}

1. What is different between default constructor vs inline field initialization?

public class Foo

{

private int x = 5;

private String[] y = new String[10];

}

public class Foo

{

private int x;

private String[] y;

public Foo()

{

x = 5;

y = new String[10];

}

}

Answer: Initialisers are executed before constructor bodies. (Which has implications if you have both initialisers and constructors, the constructor code executes second and overrides an initialised value).

1. Give the code:

Bird.java:

package com.example;

public class Bird {

private int eye;

private Integer leaf;

public Bird(int eye, Integer leaf) {

super();

System.out.println("in contructor, eye:"+this.eye);

System.out.println("in contructor, leaf:"+this.leaf);

this.eye = eye;

this.leaf = leaf;

}

public int getEye() {

return eye;

}

public void setEye(int eye) {

this.eye = eye;

}

public Integer getLeaf() {

return leaf;

}

public void setLeaf(Integer leaf) {

this.leaf = leaf;

}

}

Main.java

package com.example;

public class Main {

public static void main(String[] args) {

Bird bird = new Bird(11, 3);

System.out.println("in main, eye:"+bird.getEye());

System.out.println("in main,bird:"+bird.getLeaf());

}

}

What is the output of above code?

Answer:

in contructor, eye:0

in contructor, leaf:null

in main, eye:11

in main,bird:3

1. Give the code:

package com.example;

public class Bird {

private int eye;

private Integer leaf;

public Bird() {

super();

}

public int getEye() {

return eye;

}

public void setEye(int eye) {

this.eye = eye;

}

public Integer getLeaf() {

return leaf;

}

public void setLeaf(Integer leaf) {

this.leaf = leaf;

}

}

package com.example;

public class Main {

public static void main(String[] args) {

Bird bird = new Bird();

System.out.println("in main, eye:"+bird.getEye());

System.out.println("in main,bird:"+bird.getLeaf());

}

}

What is the output of above code ?

Answer:

in main, eye:0

in main,bird:null

1. What is the default initialization of an array in Java?
2. Give the code ?

package com.example;

public class Bird {

private int []eye = new int[2];

private Integer []huska = new Integer[2];

private boolean []lala = new boolean[2];

private Boolean []lalala = new Boolean[2];

public Bird() {

super();

}

public void showIntArray(){

System.out.println("int Array");

for (int i = 0; i < eye.length; i++) {

System.out.println(eye[0]);

}

}

public void showIntegerArray(){

System.out.println("Integer Array");

for (int i = 0; i < huska.length; i++) {

System.out.println(huska[i]);

}

}

public void showBooleanPrimitiveArray(){

System.out.println("boolean Array");

for (int i = 0; i < lala.length; i++) {

System.out.println(lala[i]);

}

}

public void showBooleanArray(){

System.out.println("Boolean Array");

for (int i = 0; i < lalala.length; i++) {

System.out.println(lalala[i]);

}

}

}

package com.example;

public class Main {

public static void main(String[] args) {

Bird bird = new Bird();

bird.showIntArray();

bird.showIntegerArray();

bird.showBooleanPrimitiveArray();

bird.showBooleanArray();

}

}

Answer:

int Array

0

0

Integer Array

null

null

boolean Array

false

false

Boolean Array

null

null

1. Why cannot reduce visibility of method inherited method from parent ?

The access modifier (§6.6) of an overriding or hiding method must provide at least as much access as the overridden or hidden method, or a compile-time error occurs. In more detail:

If the overridden or hidden method is public, then the overriding or hiding method must be public; otherwise, a compile-time error occurs.

If the overridden or hidden method is protected, then the overriding or hiding method must be protected or public; otherwise, a compile-time error occurs.

If the overridden or hidden method has default (package) access, then the overriding or hiding method must not be private; otherwise, a compile-time error occurs.

Note that a private method cannot be hidden or overridden in the technical sense of those terms. This means that a subclass can declare a method with the same signature as a private method in one of its superclasses, and there is no requirement that the return type or throws clause of such a method bear any relationship to those of the private method in the superclass.

1. Differences in boolean operators: & vs && and | vs || in Java ?

Answer:

int a = 6; // 110

int b = 4; // 100

// Bitwise AND

int c = a & b;

// 110

// & 100

// -----

// 100

// Bitwise OR

int d = a | b;

// 110

// | 100

// -----

// 110

System.out.println(c); // 4

System.out.println(d); // 6

& is bitwise. && is logical.

& evaluates both sides of the operation.

&& evaluates the left side of the operation, if it's true, it continues and evaluates the right side.

1. What is Model–view–presenter ?

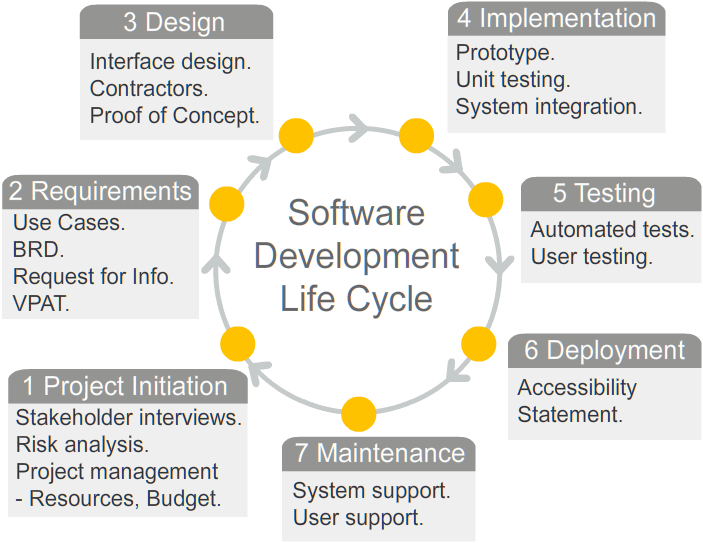
Model–view–presenter (MVP) is a derivation of the model–view–controller (MVC) architectural pattern, and is used mostly for building user interfaces.

The model is an interface defining the data to be displayed or otherwise acted upon in the user interface.

The view is a passive interface that displays data (the model) and routes user commands (events) to the presenter to act upon that data.

The presenter acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.

1. Insertion Sort
2. Design pattern
3. Quy trình phát triển một phần mềm (software-life-cycle)



1. SCUM
2. Kiểm thử phần mềm

.