

Exam. Midterm Review

1. Concept

Class . Object . Inheritance

Polymorphism . Abstraction . Encapsulation .

∴ ~~the~~ specification → implementation
data & method Encapsulated

∴ Data .

Class (global) data .

static .

不通过 instantiation .

Single copy associated only with class
∴ not obj .

Method operate on member data .

∴ static data → linked with class

non-static data → linked with each obj

Data Abstraction .

Data Hiding

public

private 类中调用

protected (package private) 可在一个包中调用

Functionality Hiding

Interface

Abstract

↓ →

Polymorphism (多态)

- Inheritance

↳ super class

sub - class

Overloaded Methods

Same name, diff signature . static Polymorphism

Overridden Methods (@Override)

Run-time polymorphism

Sub-class override method in parent base to customize method

L2. Java OOP

class are specified. 指定说明的

use dot (.) to access (static or object instantiate of) members defined in class

extends

implements 实现 extends

Concrete class

Declared Fully implemented methods

Can be instantiated to create objects

Abstract class

Declared members: public, private or protected
partially implemented

contain 1 or more abstract method

Must be extended

X be instantiated

Data declaration

```
[static] [final] [public | protected | private]  
type name [initializer];
```

Reference types.

class; passed by reference

Heap memory allocation

Automatic Garbage Collection.

Never to free heap allocation

Non-static object instantiated required.

Default: 对每个单独的对象独立

指针为静态对象; 在堆中创建对象

Static: 单独的实例数据

只与类有关系

不随对象创建

Whole program scope

Final: Constant value

Can not be overridden by inheritance

Class

Public: All access

Protected: ✓ classes in package, sub-class
✗ classes outside package

Default:

✓ class within package

X sub-class outside class

Private:

class access only

X class within package

X sub class

X outside class

Constructor

Special Method used to instantiate obj.

X method type

provide static Polymorphism

Overload → Static Polymorphism

Override → Run-time Polymorphism

Data Hiding → Access Modifier

Functionality hiding

Abstract method as API

Interface as API

Java type .

boolean, char, short, int, float, long, double.

堆内存分配

Object & use reference use heap

Java → Strongly Typed . All variables have types..

variable holds reference .. to actual object allocation on heap.

Java class

Static : Class global variable

Object instance variable

use key words : final

to disable function overriding

4. To Stream

Range Loop

```
for (type item: container)
```

Iterator Loop

```
Iterator<Integer> it = numbers.iterator();
```

```
while (it.hasNext()) {
```

```
    S.out.println(it.next());
```

```
}
```

ListIterator Loop

```
ListIterator<Integer> It = numbers.listIterator();
```

```
while (It.hasNext()) {
```

```
    SOP (It.next());
```

```
}
```

★ SOLID

Single Responsibility #

Open-Closed.

Liskov Substitution

Interface Segregation (接口隔离) Principle

Dependency Inversion 独立反向

S: each class \rightarrow only one responsibility or purpose

使用封装性.

1. 所有与 purpose 相关的数据均为 private
2. 提供 public API

O: class is open to extension.

//: new sub-class & polymorphism

class is closed to be modified.

Once tested, use class as a Superclass

L: Liskov Substitution Principle

Is-A: Any subclass may be substituted for its
Super class

子类可以替换父类

Run time - Poly

Is: Interface Segregation

No class should depend on any method
it doesn't implement (use)

Dependency Inversion Principle

Loosely coupled Design

Depend on abstractions

Functionality Hiding: Subclass is never named

X clear call new or static.

String:

```
String s1 = "cat";
```

```
char a[] = {'c', 'a', 't'};
```

```
String s2 = new String(a);
```

String is not array
of char

15. Java containers

Interface \rightarrow ordered collections.

List <T> name = new ~~ArrayList~~ Vector <T> ();

Vector ^{同步的} Synchronized, 线程安全的数据结构

ArrayList (Unsynchronized Vector)

LinkedList: 链表

Queue: (e.g. stack)

LIFO

Deque (e.g. queue)

FIFO

排序

nature order, 需要 comparable 接口

Collection.sort(list);

list.sort(cmp);

comparator

注意: 使用 comparator
时 implements
comparator

L6


Early & eager initialization

1. 实时
2. 资源占用 (计算机主内存空间)
3. 可能存在空间浪费

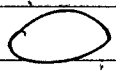
Late & Lazy initialization

1. 内存分配当 requested ..
预留空间
2. 非实时生效

Data Hiding

```
public static synchronized  getInstance () {
```

```
    if (null == instance) {
```

```
        instance = new  ();
```

```
    }
```

```
    return instance;
```

```
}
```

lambda.

names.stream().filter(s -> s.startsWith("clm"))

.forEach(System.out::print)

Lambda: 一种匿名函数

将功能作为参数 -> 另一个method

可以将代码拆成数据

Syntax: 语法

() -> {}

注意理解: 将功能作为参数

example

在A中
printValue(a):

for (String a : listA) {

A.printValue(a);

a1. forEach(x -> A.accept, printValue());

a2. forEach(A::printValue);

CS <String> method P = A::printValue;

a3. forEach(x -> method P, accept(x));

thread. (线程类)

如需在多线程中插入其他线程，则可引用 Thread 类
类中有 start 方法。

用 Lambda as in-line implementation of a Functional Interface.

use ~~in the~~ inner class 各种代码见 PPT.

```
List<String> states =  
    states.  
    stream().  
    .filter(s -> s.startsWith("u"))  
    .map(String::toLowerCase)  
    .sorted()  
    .forEach(s -> System.out.print(s + " "));
```

Method Reference

Some class :: a Static Method 静态

myObject :: an Instance Method : 一般对像的方法指针

Some class :: new :: constructor.

Anonymous Inner class.

空在一行里，完整 specification 在 new.

Java Inner class.

```
public class OuterClass {
```

```
    private interface InnerHelperInterface {
```

```
    }
```

```
}
```

注意: 内部类中

内部接口不被定义为一个内部类

外部类始终为 public,

内部类为 public, protected or private (not default).

In:

```
public class OuterClass {
```

```
    private class InnerHelperClass {
```

```
    }
```

```
}
```

若为静态类则实例化属于外部类

若为非静态, 则内部类初始化为 object instance:

```
OuterClass oc = new OuterClass();
```

```
InnerClass inc = oc.new InnerClass();
```

Inner Class

static methods need ^{inner} Class also be static.

Non-static Method belongs to each instance.

匿名内部类

Anonymous Inner Class

In-Line unnamed class specification

被提为函数做接口

一般为实现接口的一种形式

eg:

```
student.sort(new Comparator<Student>() {
```

@Override

```
public int compare(Student s1, Student s2) {
```

```
return s1.getAge().compareTo(s2.getAge());
```

```
}
```

```
}
```

```
);
```

Java. Auto Boxing

自动在某些基本数据类型和其对应的对象包装类之间切换。

Character \leftrightarrow char

Integer \leftrightarrow int

Double \leftrightarrow double

★ String \leftrightarrow String?

Difference

eg:

Integer number \Rightarrow

int num \Rightarrow

- | | |
|---|---------------------|
| 1. Reference type | 1. primitive type |
| 2. Class needs to be instantiated or set to null. | 2. no instantiation |
| 3. Object has more useful members | 3. no any members |
| 4. Requires Additional Storage. | 4. less storage |
| | 5. Fast Access |

Java interface

Reference type

Contains:

Data:

public static final (constant in memory).

Methods :

All Methods 隐式公开.

public 关键字的用途

public interface can be accessed to all class

otherwise only accessible to classes in same package.

interface can be extends by other interface

private : only available to package

All methods are public

Defaut & sentre methods with implementation

eg:

eg:

1. Comparable \rightarrow `Collections.sort(List)` // natural order
2. Comparator `Collections.sort(List, comparator)`
Override `compare()`
3. Runnable

Lambda

previous review

Imperative programming \rightarrow 必修的, 强制性 \rightarrow Low

Declarative Programming \rightarrow High Level
(Functional Programming) Define "What's" without "How"

System.out.println(" " + names.stream().filter(s \rightarrow

s == "dan").forEach(System.out::print);

" \rightarrow " Java 8 新特性 : NO

借鉴其他编程语言而来: Lisp, Clojure, Erlang, Ruby.

可以存在于其他 JVM language 如 Groovy, Scala.

Lambda "标识符" 可看作数据

() \rightarrow { }

List of parameter

可省略数据类型

可有吗? () "当只有一个参数"

ps: "Arrow token"



"{ }"

Single expression

1. 不需要分号
2. 不需要大括号
3. "return" 关键字省略

eg: x()

p
(x, y)
int x, int y;

eg:

Lambda expression

Arrays.sort (Array, (a,b) → Person.compareByAge (a,b));

Method Reference

Arrays.sort (array, Person::compareByAge);

using in-line implementation with Lambda

Class Def

@FunctionalInterface

public interface CreateDivide {

int divide (int t1, int t2);

}

```
public void simpleLambda() {
```

```
    CreateDivide intDivide = (int x, int y) -> x/y;
```

```
    System.out.println(intDivide.divide(2, 3));
```

```
}
```

```
}
```

匿名内部类 多线程能在 lambda 中的使用

```
void RunnableAnonymous() {
```

```
    Runnable rAnonymous = new Runnable() {
```

```
        @Override
```

```
        public void run() {
```

```
            // body
```

```
        }
```

```
    };
```

```
    Thread t = new Thread(rAnonymous);
```

```
    t.start();
```

```
}
```

另一种写法

```
Runnable r = () -> System.out.println("——");
```

```
Thread t = new Thread(r);
```

```
t.start();
```

~~use~~ use in-Line

```
void runnableLambda() {
```

```
    Thread t = new Thread(() -> Say(" "));
```

```
    t.start();
```

```
}
```

Method Reference

Static Ref.

SomeClass :: aStaticMethod

Non-static

myObject :: an Instantiate Method

Constructor:

SomeClass :: new

Conclusion

Lambda

" \rightarrow " 的使用是创建一个匿名方法并可以被当作数据传递。

Po: instanceof 一个: 无意义

Interface Predicate<T>

boolean test(T t)

三个方法 and or negate

对应 && || !

参见 www.cnblogs.com/rever/p/9773743.html

Function 接口

1. 泛型类 <T>

2. 通配符

www.cnblogs.com/rever/p/9725173.html

Swing 界面

JComponent Base class for all J* class

除 JFrame, JDialog, JApplet

线程不安全

有内容卡顿

Exception

1. Exceptional Event
2. Error or Fault
3. Disruptive
4. Maybe unrecoverable

所有的异常均为 Checked Exception.

除 Error & Runtime Exception
及其子类

可预计的错误条件.

File I/O with user specified filename

外部 Error like Handname Error

内部 : bug.

Runtime Exception
NullPointerException

Caught & specified.

Throw 抛出异常.

try // 可能抛出异常的代码段

{ catch // 当异常被抛出时执行

{ finally // 总是执行

}

Thread & Process

Concept 1: Concurrent Software

Software allow users to do more than one things at a time.

Concept 2: Process

a self-contained execution environment.

Concept 3: Thread

1: 线程 implements Runnable 接口

2: 需创建子类继承 Thread (使用)

1. 继承 Thread 类 (或使用 Runnable 接口)

2. Override run 函数

3. new Thread() 实例化一个线程 多种多线程写法

4. `Instantiation.start()`;

多线程程序可访问相同的数据

1. 读数据 并行

0. 写数据 可能会造成错误, 需加锁进行.