雨课堂知识点总结（一）

0-1 试试 Java
第一题
Integer a = new Integer(3);
Integer b = 3;
int c = 3;
System.out.println(a == b);
System.out.println(a == c);
控制台输出结果是：
False；True
解析：
针对第四行，a 和 b 是两个不同的 object， 在 heap 中指向不同的地址，“==”判定对象等价性（将在 3.5 节继续讨论）。
这对第五行，a 作为一个 Integer 对象， 将会首先被 auto-unboxing 为一个 int，然后再跟 c 比较。针对 int 类型的==，比 较的是值。
第二题
String a = “c”;
String b = “c”;
System.out.println("a and b: " + a == b);
控制台的输出结果是：
False
解析：
+的优先级比==要高，先计算两个字符串+，然后再跟 b 比较。
第三题
public static void main(String args[]){
System.out.println(2.00 - 1.10);
}
控制台输出结果是：
0.8999999999999999
第四题
List<Integer> list = new ArrayList<>();
for(int i = -3; i < 3; i++)
list.add(i);
for(int i = 0; i < 3; i++)
list.remove(i);
System.out.println(list);
解析：
[-2, 0, 2]
第五题
String s = " Hello";
s += " world ";
s.trim();

System.out.println(s);
解析：
" Hello World "

雨课堂知识点总结（二）

1.1 软件构造的多维度视图
第一题
Memory dump 属于软件三维度视图的：
Moment 和 component-level view
第二题
Execution stack trace 和 code snapshot 在软件三维度视图中的共性特征是：
都是 run-time view
第三题
Code Churn 和 AST 分别是 Period，Build-time 的视图
第四题
Static linking 和 Dynamic linking 的区别在于：
前者发生在构造阶段，后者发生在运行阶段
第五题
以下说法正确的是：
Code static analysis 是发生的在 build-time
Deployment 是把 build-time 的软件转换为 run-time 的软件的手段之一
Files 随时间发生变化，产生各个不同版本，按时间连起来形成 period view
对软件的 profiling 和 tracing 均发生在 run-time

雨课堂知识点总结（三）

1.2 软件构造的质量指标

1.软件构造的 external quality factors:

correctness 正确性

extendibility 可扩展性

reusability 可复用性

ease of use 易用性

2.关于软件构造的质量指标：

健壮性刻画了软件能够恰当的处理 spec 范围之外的各类异常情况的能力。

代码行数 Loc 是内部质量指标之一，但它可能对多项外部质量指标产生影响。

程序的可复用性与程序的开发代价/运行效率之间存在折中。

3.关于 correctness 和 robustness 的区别：

前者是针对需求的正确实现，后者是针对需求之外的其他情况的恰当实现。

4.LoC 和 code complexity 很高，并不代表一定有很差的 reusability 和 extendibility。

5.对代码的时间/空间复杂度进行优化，可能带来其他 external quality factors 的降低。

6.每向软件里增加一点功能，都要确保其他质量属性不受到损失。

雨课堂知识点总结（四）

2.1 软件过程与配置管理

1.agile development 敏捷开发过程的特征

增量式过程

迭代过程

测试驱动开发（Test-Driven）

持续集成，持续交付

V 字模型（确认/验证）

2.关于软件配置管理 SCM 的说法正确的是：

用于追踪和控制软件开发过程中的变化

其基本管理单元是软件配置项 SCI，即开发过程中发生变化的基本单元

版本是为软件处于特定时刻（moment）的形态指派一个唯一的编号

git 是分布式版本控制系统

软件配置项 SCI 是软件演化过程中发生变化和 SCM 管理变化的基本单元，不需再细分

3.关于 git 说法正确的：

git 中在本地机器上的.git 目录对应于 SCM 中的配置管理数据库 CMDB

git 中的 SCI 是文件，有三种形态：已提交（modified）已暂存（staged）已提交（committed）

4.用于将 github 上的某个 git 仓库设置为本地仓库的远程仓库的指令是：

git remote add

用于将当前 staging area 中的文件写入 git 仓库的指令是：

git commit

5.针对 git 仓库的 object graph，正确的说法是：

一个 branch（分支）本质上相当于一个指定特定 commit 节点的指针

可以有两个不同的 branch 指向同一个 commit 节点

git commit 指令相当于在 object graph 当前分支 HEAD 指向的 commit 基础上，派生出一个新的 commit 节点

6.针对 git 中 commit 节点中数据结构，说法正确的：

若说 commit 相比其他 parent 来说，某文件 f 未发生变化，则 f 在.git 中不会重复存储。

如果某个 commit 节点仅存在于远程服务器的 object graph，那么当本地向远程 git push 的时候，会出现错误提示

包含一个 tree，tree 中包含一组指针，指向本次 commit 中包含的所有文件

7.将本地仓库中 master 分支的最新提交推送至远程仓库的指令：

git fetch origin master

git merge

git push origin master

其中 origin 是远程仓库的网址

雨课堂知识点总结（五）

2.2 软件构造工具
1.用于软件设计阶段描述设计思想和设计结果：
Modeling languages(e.g.,JSON)
2.关于软件构造过程各阶段的说法不正确的是：
Profiling 是 static analysis 的一种典型形式
正确的有：
Code review 的目的是发现代码中的潜在错误
Refactoring 是在不改变代码功能的前提下重写代码，以消除 bug，提高质量
Build 是将软件从开发状态转化为可运行状态的过程
3.以下环节无需执行正在开发的软件：
Code review

需要：
Dynamic code analysis
Debug
Testing
4.Dynamic code analysis/profiling 解决不了的问题是：
发现程序中潜在的重复代码以便于抽取出来形成可复用函数/类
可以：
发现程序运行过程中的内存分配和占用情况
发现程序运行过程中每个类被实例化的数目，及其所占用的内存
发现程序潜在的性能瓶颈
5.以下过程可纳入自动化 build：
Compiling .java into .class
Executing JUnit test cases
Using Checkstyle tool to check if code follows Google's Java code style
Packaging .class files into .jar file and deploying it to a remote server
6.常规的构造次序是：
programming-refactoring-debugging-testing-dynamic code analysis/profiling-code review static code analysis-build
通过 code review 和 profiling 找出可能的 bug，通过 testing 找出真实的 bug，通过 debug 找出 bug 的根源
先根据 spec 构造完备的测试用例，后续对代码的任何修改，都应重新运行测试用例
Build 脚本是由配置语言书写，告知 build 工具如何一步一步完成自动化 build 任务

雨课堂知识点总结（六）

test

1.要为某个方法 A m(int b,String c)构造黑盒测试用例，那么设计实现 Junit 测试用例不需要依据的内容为

m()的内部实现代码

需要依照的内容包括：

m()的 pre-condition（该方法输入参数满足的条件）

m()的 post-condition（该方法执行后返回值满足的条件）

类 A 的等价性判断方法 A.equals()

2.并不是说测试用例的数量越多，越容易发现潜在的 bug

再好的测试也无法证明代码里 100%不存在 bug

设计测试用例时，需要给出输入数据和期望的输出结果

测试用例具有优先级，应该将最容易发现错误的用例先执行

TDD 的思想是：先设计方法的 spec，然后根据 spec 设计测试用例，然后再写代码并确保代码通过测试

3.all are good time to re-run all your Junit tests

Before doing git add/commit/push

After rewriting a function to make it faster

when using a code coverage tool such as EclEmma

After you think you fixed a bug

4.在使用等价类划分方法进行测试用例设计时：

如果输入参数 a 在 spec 中仅被规定 a>10 且未说明 a<=10 应该如何，那么无需测试 a<=10 的情况

采用笛卡尔积“全覆盖”策略进行测试用例设计，会导致用例数量多，测试代价高

采用“覆盖每个取值”的策略进行用例设计，测试代价低，但测试覆盖度可能较差

5.all are useful for choosing test cases in test-first programming,before any code is written

Black box 黑盒测试

partitioning 等价类划分

Boundaries 边界值

6.以下说法不正确的是：

如果某个 bug 已被正确修复并已经通过测试，那么为了降低后续测试的代价，应将该 bug 对应的测试用例从测试库中删除

正确的：

如果发现了一个新的 bug，需要返回到版本仓库中对之前的各个版本进行测试，以确认该 bug 最早是在哪个历史版本中引入的

代码覆盖度 code coverage 是指所有测试用例执行后有多大百分比覆盖了被测程序的所有的代码行

可以从被测代码中寻找依据来设计处于“边界”上的测试用例

7.关于 JUnit 的说法不正确的是：

如果一个 Java 测试类定义了多个@Test 方法，那么它们按照在代码中出现的先后次序加以执行

正确的有：

某方法前标注着@Test，意味着它是一个测试方法。@Test 是 Java 中的 annotation

如果未通过测试，方法中的 assertXXX()将抛出 AssertionError

一个 Java 测试类可以定义全局属性并在@Before 方法中对属性进行数据准备，在@Test 方法中使用数据

雨课堂知识点总结（七）

3.1

1.Java 中的 Primitive Type（int,char,boolean 等）和 Object Type（String,Boolean,Calendar）的差异是

前者在 Stack 中分配内存，后者在 heap 中分配内存

使用前者的时空代价低，使用后者的时空代价高

前者和后者中的某些类型可通过 auto-boxing 进行自动转换，例如 int 和 Integer

2.Static type checking 和 dynamic type checking 的区别是

前者在编译阶段发生，后者在运行阶段发生

前者比后者更能带来程序的健壮性，因为可以在程序投入运行前就发现错误

前者是关于“类型合法性”的检查，后者是关于“值的合法性”的检查

3.下面的代码会在运行阶段发生错误:

```
int [] arr = new int [] {1,2};
arr[2] = 3;
```

以及

```
String s = null;
System.out.println(s.length());
```

4.static type checking 所能处理的错误包括：

所调用函数的参数数目错误

函数的 return 语句返回的变量类型与函数声明中的返回值类型不匹配

赋值语句右侧的值类型与左侧的变量类型不匹配

5.不具备 static typing 的能力的语言包括：

Python Ruby Perl PHP

6.
```
String a = "5"+6;
System.out.println(a);
```

上面代码会在控制台打印 56

```
int a = "5"+6;
System.out.println(a);
```

此处会发生编译错误

雨课堂知识点总结（八）

3.1（B）

1,关于 mutable 和 immutable 的说法正确的是：

所有的简单数据类型和所有相对应的封装类（Integer,Double,Boolean 等）都是 immutable 的

所有数组都是 mutable 的

使用 immutable 类型可以降低程序蕴含 bug 的风险，但其时空性能相对较差

2,针对 final 关键字，说法正确的是：

A final class declaration means it cannot be inherited

A final variable means it always contains the same value/reference but cannot be changed

A final method means it cannot be overridden by subclasses

3.String a = "a";

String c = a;

a += "b";

c += "c";

StringBuilder b = new StringBuilder(a);

StringBuilder d = b;

b.append("b");

d.append("c");

假设执行之后未进行任何垃圾回收，此时内存里共有 3 个 String 对象和 1 个 StringBuilder 对象

此时 c 的取值是 ac,d 的取值是 abbc

4,关于 immutable 和 mutable data type 的说法，正确的是：

使用不可变类型，对其频繁修改会产生大量的临时拷贝

可变类型可直接修改其值而无需太多拷贝，从而提高效率

不可变数据类型更安全，因为其值无法修改

使用可变类型做全局变量，可在不同模块之间高效率的进行共享数据读写

5,final List<String> l1 = new ArrayList<>();

List <String> l2 = new ArrayList<>();

1.l1.add("a");

2.l1.set(0,"b");

3.l1 = l2;

4.l2 = l1;

上面无法通过 static type checking 的是 3

因为 l1 为 final

6.List <String> k = new ArrayList<>();

k.add("lab1 ends");

Iterator it = k.iterator();

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

it.next();

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

k.remove(0);

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

输出的结果为 true false true

第三个在迭代器迭代过程中用 remove 删除的话，以后随便输出信息，所以为 true

7.List<String> t = new ArrayList<>();

t.addAll(Arrays.asList("a","b"));

Iterator<String> i = t.itertor();

while(i.hasNext())

    if(i.next()=="a")

        i.remove();

期望结果是 t 中只包含"b"

下面说法正确的是：

正常执行，结果与期望一致！

雨课堂知识点总结（九）

3.2 Specification

1.两个方法具有“行为等价性(behavior equivalence)”：

站在客户端的角度看，它们实现相同的功能

站在客户端的角度看，它们可能展现出不同的性能

它们具有相同的规约(spec)

其实是针对同一个 spec 来说是等价的。若对这个 spec 进行更改，这两个方法也许就不等价了

2.关于方法 spec 的说法：

程序员针对给定的 spec 写代码，需做到“若前置条件满足，则后置条件必须要满足”

前置条件是对 client 端的约束，后置条件是对开发者的约束

若客户端传递进来的参数不满足前置条件，则方法可直接退出或随意返回一个结果

3.在 Java 的语法中，使用@param 表达一个方法的 pre-condition，使用@return 和@throws 表达方式的 post-condition

4.除非在 post-condition 中明确声明过，否则方法内部代码不应该改变输入参数

方法的 spec 描述里不能使用内部代码中的局部变量或该方法所在类的 private 属性

若在方法的 post-condition 中声明“client 端不能修改该方法所返回的变量”，不能减少该方法的潜在 bug

若为某方法设计 JUnit test case，在任何 test case 中对该方法的调用必须遵循其 pre-condition

5.如果修改了某个方法的 spec 使之变弱了，那么可能发生的是：

client 调用该方法的代价变大了，即 client 需要对调用时传入该方法的参数做更多的检查

程序员实现该 spec 的难度降低了，自由度增加了

如果使用椭圆面积表示 spec 的强度，那么该方法的椭圆面积增大了

该 spec 的实现方式变多了

6.spec 的强度：

前置越强 spec 越弱，后置越强 spec 越强，后置比较需要在相同的前置条件下。若前置后置都更强则无法比较

雨课堂知识点总结（十）

3.3ADT

1.类 WordList 有四个方法，根据其方法定义来确定其类型

《1》public WordList(List<String> words)

《2》public void unique()

《3》public WordList getCaptitalized()

《4》public Map<String ,Integer> getFrequencies()

使用 C,M,P,O 分别表示 Creator 构造器,Mutator 变值器,Producer 生产器,Observer 观察器

1->C

2->M

3->P

4->O

2.下面关于 ADT 的 RI 和 AF 正确的是

ADT 的 Abstract 空间（A）中的某个值，在其 Rep 空间（R）中可能有多个值和其对应

若 ADT 的某个方法返回一个 mutable 的对象，并不一定表明该 ADT 产生了表示泄露（defensive copy）

一个 immutable 的 ADT，其 rep 可以是 mutable 的

3.关于 invariants,AF 和 RI，说法正确的是

如果一个 immutable 的 ADT 存在 rep exposure，那么就违反了该 ADT 的 invariants

如果在一个 mutator 方法内没有 checkRep()，那么 RI 就可能被违反了

两个 ADT 有相同的 rep 和相同的 RI，但可能 AF 不同

4.用于检查 ADT 的 Rep Invariants 是否保持为真的 checkRep()最好应该以下类型的方法结束前调用

Mutator

Creator

Producer

Observer

5.class{

private String s;

private String t;

}

它的可能 Rep Invariant 会是

s contains only letters

s.length()==t.length()

s is the reverse of t

6.ADT 的某个方法的 spec 需要以注释的形式放在代码中，在撰写这部分 spec 的时候，不能用到的信息是：
Mathematical values in the Rep space(R)
可以使用：
Parameters of operation
Data type of return values of the operation
Exceptions thrown by the operation
Mathematical values in the Abstract space(A)
解析：
Spec 要给 client 看，那么所有内部的东西都不能用。R 是 rep 的值空间，只能开发者自己了解。
7.在对 ADT 的方法进行 JUnit 测试时，以下说法正确的是：
对 constructor 方法，测试用例中需要构造新对象之后调用 observer 方法确认构造结果是否正确
对 observer 方法，测试用例中需要使用其他三类方法构造一个对象，再执行该方法并判断结果是否正确
不正确：
如果某方法的返回值为 void，则无法为其撰写测试用例，因为无法 assertEquals()
对 mutator 方法，测试用例中需要在该方法执行之后调用 producer 方法确认是否做了正确的 mutate
8.不正确：
只要有非 final 的 field，就一定产生表示泄露
除了初始化，immutable 的类一定不能存在其他任何改变 rep 的方法
正确：
只要有 public 的 field，就一定有表示泄露
checkRep()方法可能消耗大量运算在程序投入实际运行的时候要注释掉
解析：
所以不到迫不得已千万不要用 public
final 是来支持 immutable 的，与是否存在表示泄露无直接关系
可以有 beneficent mutation
assert 语句在投入真正运行的程序中是没有意义的，需要注释掉。前提是开发者利用 checkRep()已经发现了所有违法 RI 的 bug 并修复了
9.针对你设计的一个 ADT，不应该提供给 client 看的内容包括：
AF
RI
Rep exposure
Testing strategy
Rep
Implementation
Test cases

提供：
Spec

雨课堂知识点总结（十一）

3.4Object-Oriented Programming(OOP)

1.关于 static 和 final 的说法，正确的是：
static 类型的方法，调用时无需创建类的实例对象，可直接通过类名使用
被声明为 final 类型的类，无法从中派生出子类
被声明为 final 类型的方法，无法在子类中被 override
类 A 的 static 方法中不能直接调用 A 的 instance 方法（而是要 new 一个 A 的对象再调用）；A 的 instance 方法中可以直接调用 A 的 static 方法
不正确：
一个变量被声明为 final，意味着它在被首次赋值之后，其内容就不能再变化
2.关于 Java interface 的说法，正确的是：
不能有 constructor（构造方法）
不能有 final 方法
不能有 private 方法
可以有 static 方法
可以有 fields（属性）
解析：
可以有属性的，都是 public static final 的
3.关于 class 和 interface 的说法，不正确的是：
一个类只能 implements 一个接口
一个类不能同时 extends 另一个类和 implements 一个接口
正确的：
一个接口可以 extends 一个或多个其他接口
一个类 implements 了一个接口，意味着它必须要实现该接口中的所有方法
一个类除了实现其 implements 的接口中的方法，还可以增加新的方法
4.某方法的定义是 public int getLength (List <String> list, boolean bFliter)，对该方法合法的重载：
private int getLength (List<String> list, String regex)
public Integer getLength (List<String> list)
public int getLength (List<String> list) throws IOException
不合法的：
public void getLength (List<Object> list, boolean bFliter)

5.关于 Java OOP 中 override 和 overload 的异同，说法正确的是：

前者的参数列表不能改变，后者的参数列表必须发生变化

前者的返回值类型不可变化，后者的返回值类型可以变化

前者的类型检查发生在 run-time，后者的类型检查发生在 compile-time

在子类里 override 父类方法时，@override 不是必须的

6.class Car{

public String refuel()

{ return "R";}

}

Class Tesla extends Car{

public String refuel(){return "C";}

public String refuel(double price)

{return "p";}

}

无法通过 static type checking 的是

Car c = new Car();

c.refuel(10);

Car c = new Tesla();

c.refuel(10);

7.

class Car{

public String refuel()

{ return "R";}

}

Class Tesla extends Car{

public String refuel(){return "C";}

public String refuel(double price)

{return "p";}

}

能获得内容为"C"的字符串是

Car c = new Tesla();

c.refuel();

Tesla t = Tesla();

t.refuel();

8.类 A 和 B，BextendsA，二者分别有一个 apply（A a）方法，具有不同的返回值类型

假如

A a = new A();

B b = new B();

下面正确的是：

b.apply(a)与（（B）b）.apply((B)a)不等价

注意：将 a 向下强转做不到，运行时候会抛出异常 ClassCastExcepion

((A) b).apply(a) 与 b.apply(a)是等价的

因为 B 是 A 的子类，所以 b 必然是 A 的实例，所以将 b 转成(A)b,本质上还是等价的

9.关于 OOPpolymorphism(多态）的各选项中：

generics 和 overriding 不是同义词

下面的是同义词：

subtype polymorpgism 和 inclusion polymorphism

Parametric polymorphism 和 generics

ad hoc polymorphism 和 function overloading

雨课堂知识点总结（十二）

3.5 Equality

1.

ADT 的 equals( )需要满足的三个性质是[填空 1]性、[填空 2]性、 [填空 3]性

答案： 自反性;对称性;传递性;

2.

以下针对 ADT 等价性的说法,不正确的是

A

如果对象 a 和 b 的 R 值被 AF 映射到相同的 A 值，则 a 和 b 等价

B

对对象 a 和 b 调用任何相同的方法，都会得到相同的返回值,则它们是等价的

C

对象 a 和 b 不等价,那么该 ADT 中不应存在任何方法 op 使得 a. op()=b. op()

D

对象 a 和 b 是等价的，那么 a 和 b 的 rep 中每个 field 的值也- -定是相等的
正确答案：CD
A 选项是从 AF 的角度判断等价性。
B 选项是从外部观察者的角度来判断等价性。
C 选项:是可以存在的这样的操作的。即使这个操作返回相同的结果,该 ADT 还应有其他操作会让 a 和 b 执行后的结果不同。
D 选项:判断等价性取决于 A 值的等价性而非 R 值。很有可能多个 R 值对应于同一个 A 值。
3.
Java 中有两种不同的操作来验证对象的等价性，分别为==和 equals(),它们的”学名”是[填空 1]等价性和[填空 2]等价性。
答案： 引用 对象
4.
某个 ADT 的 REP 是:private int no;
private String name;他的 AF 是:
AF(no) == ID of a student
那么以下 equals( )正确的是：
A： return this.no == that.no
B:return this.no == that.no&&this.name == that.name
C:return this.no == that.no&&this.name.equals(that.name)
D:return this.name.equals(that.name)
答案：A
5.
针对 mutable 的 Java 类，有两种等价性，分别为[填空 1]等价性和[填空 2]等价性。
答案：观察 行为
6.
以下关于的等价性的说法，不正确的是
A:0bject 类缺省的 equals()是判断两个对象内存地址是否相等

B:若 a 和 b 满足 a. equals(b)为真，那么 a. hashCode()==b. hashCode( )也应为真

C:若 a. equals(b)为假，那么 a.hashCode( )==b. hashCode()可以为真

D:针对 mutable 的类，由于其 rep 在变化，直接使用==判断是否相等即可，无需 override equals( )

正确答案：D
D 选项:这要取决于实际需求。例如 Java 中 List 和 Date 等 mutable 的类,都 override 了 equals()方法,实现观察等价性。
7.
关于 hashCode()的说法，不正确的是_
A:具有相同 hashCode( )返回值的两个对象 a 和 b,必须做到 a . equals(b)为真
B:若 hashCode( )始终返回同一个值， 则违反了 0bject 对 hashCode( )的要求
C:如果不 override hashCode( )方法，object 类缺省实现也可以满足要求

D:一个对象实例的 hashCode( )返回值,在其生命周期内可以发生变化
正确答案: BCD
答案解析:
针对 C 选项:还是要看需求是什么,不能一概而论
针对 D 选项: mutable 对象的
hashCode 在其 mutator 方法执行后，就会发生变化,因此在 hash table 中的位置也可能随之发生变化。
8.
Date d = new Date(2019,4,1);
Set<Date> hs = new HashSet<>();
Set<Date> ts = new TreeSet<>();
hs . add(d);
ts. add(d);
d. setYear(2020) ;
println(hs. contains(d));println(ts . contains(d));
该代码执行后会打印出：
A： True; True
B： True; False
C： False; True
D： False; False
正确答案: C
答案解析: :
hashCode ,顾名思义，mutable 对象的 hashCode 变化之后，只会影响受 hashSet ,不会影响 TreeSet
9.
Date d = new Date(2019,4,1);
Date f = new Date(2019,4,1);
List<Date> al=new ArrayList<>();
List<Date> ll=new LinkedList<>();
al. add(d);
ll . add(f);
println(d.equals(f));
println(al.equals(ll));
该代码执行后输出的是：
A:True; True
B:True; False
C:False; True
D:False; False
正确答案: A
答案解析:
Date 是 mutable 的,它实现的是观察等价性,所以 d 和 f 两个值等价。

List 也是 mutable 的,实现的也是观察等价性,所以判断其中每个元素的等价性,而 d 与 f 是等价的,故 al 和 l 也是等价的。这个跟其具体的类型(ArrayList 和 LinkedList)无关。

雨课堂知识点总结（十三）

5.1 可复用性

1.

Programming for reuseProgramming with reuse 二者的区别：

for:开发可复用的软件;with:用可复用的软件开发自己的软件

for:难点在于抽象

(abstraction)，让开发出的软件能适应于不同但相似的应用场合

with:难点在于适配

(adaption),让自己的软件与来自外部的软件之间做好恰当的连接

2.

Lab2 中，你开发了 Graph<L>,然后在 FriendShipGraph 中使用 Graph<L>表示人与人之间的社交网络，此为

Module level reuse

3.

为了让你的 Lab2 具备可视化功能，你决定复用 Lab1 的 TurtleGraphics,于是在代码里加入 import turtle.*, 然后在中用 Turtle 的相关类和方法执行图的可视化，此为

Library level reuse

解析:

潜藏的操作是你必须将 turtle 的 jar 包或.class 目录加入你的项目 path 中。该 jar 表示个可复用的外部 library。就如同:为了让你的程序具备 Junit 测试能力,你必须将 junit.jar 放入你的 path 里。

4.

你在 GitHub_上搜索了某个 ConvexHull 的算法，将其代码复制到你的 Lab1 中，这属于

Code level reuse

答案解析:

虽然不算“剽窃”,但如果你的软件投入商业用途,必须要遵循对方的开源许可协议

5.

以下技术对开发高可复用性的软件有积极意义

泛型/参数化，例如 Graph<L>中的<L>

使用 interface 定义操作，而非用 class 直接实现 op

设计和实现 abstract class

使用 override 和 overload

将 ADT 的 rep 设置为 private 和 final,并避免表示泄露

精心撰写符合要求的 spec 并生成 Java Doc

6.

Framework 是一种典型的复用形态，它与传统的 API 复用存在区别，以下正确的是

API 复用是将外部开发的 API 放到自己的代码中去调用，自己的代码是可执行

程序的主体
Framework 复用是将自己的代码填充到 framework 中，可执行程序的主体是 framework
API 复用的学习周期短，framework 复用的学习周期较长
不正确的：
API 复用的粒度大，framework 复用的粒度小

雨课堂知识点总结（十四）

5.2

subtyping

1.
Behavioral subtyping 必须要满足的条件，不包括以下_
A
子类型可以增加父类型中所没有的新方法
B
子类型 override 父类型的某方法，子类型方法需具备相同或更弱的 post-condition
C
子类型必须要具备与父类型相同或更弱的 invariants(不变量)
D
子类型 override 父类型的某个方法，不能比父类型方法抛出新的异常类型，但可比父类型方法抛出的异常更少
正确答案：BC
2.
关于 Behavioral subtyping 的说法，不正确的是_
A
子类型 override 父类型某个方法，其返回值类型应该与父类型方法的返回值类型相同或者更具体(子类型)
B
子类型 override 父类型某个方法，其参数的类型应该与父类型方法的参数类型相同或者更具体
C
某个类是 immutable 的，它可以派生出一-个 mutable 的子类
D
子类型 override 父类型某个 public 方法，子类型中该方法的可见性可以为 private
答案：BCD

C 选项: java 允许你这么设计, statictype checking 也可以通过,但却是违反 LSP 的,因为 immutable 也是 invariant (不变量),子类型的

invariant 应该等于或强于父类型,而一个 mutable 的子类型就不再具备这个 invariant。

D 选项:把一个 public 方法 override 为 private 方法,那么如果用子类型的对象取代父类型对象,就无法调用这个操作了，所以无法取代，所以违反 LSP。

3.

```
public class A {
public object a (String d) {
return "";}}
public class B extends A {
@Override
public String a (Object d) {
return null;}
}
```

该段 Java 代碣是否能通过 statictype checking?

A：能

B：不能

正确答案:B

答案解析:

虽然方法 a 在类 A 和类 B 里符合 covariance 和 contra-variance ,但因为 Java 不支持 contra-variance ,这里的 @Override 是不能成立的。

4.

```
public class A {
public object a (String d) {
return "" ;}}
public class B extends A {
public String a (0bject d) {
return null;}}
```

亥段 Java 代碣是否能通过 statictype checking?

A 能

B 不能

正硝答案: A

解析：方法 a 在类 A 和类 B 里符合 covariance 和 contra-variance，但因为 java 不支持 contra-variance，于是

编译器就把 B 中的 a 方法看做 A 中 a 方法的 overload

5.

```
void print(List<Object> list)
{...}
```

以下__作为参数传递进去不是对它的合法调用？

A： List<Integer> a;

B List<?> a;

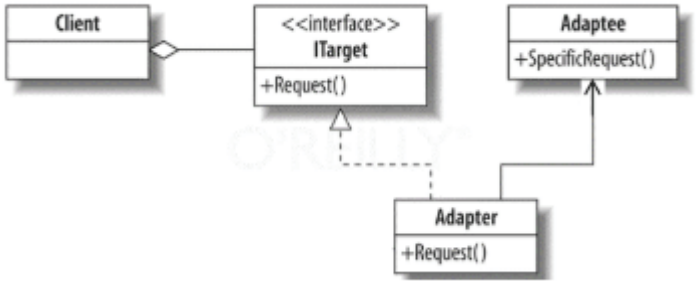
C： Arraylist<Integer> a;

D： List<? extends Object> a;

E： List<Object> a;
答案：abcd
6.
void print
(List<? extends Number> list)
{...}
以下__作为参数传递进去不是对它的合法调用？
A： List<Integer> a;
B: List<?> a;
C: ArrayList<Integer> a;
D: List<? extends Integer> a;
E: List<0bject> a;
正确答案：BE

雨课堂知识点总结（十五）

1.

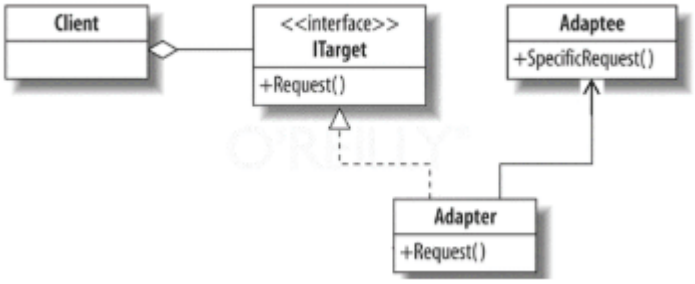


关于 Adapter 模式的说法，正确的是__

A：Adapter 类可提供被复用的方法，但与 Client 要求的 spec 不吻合

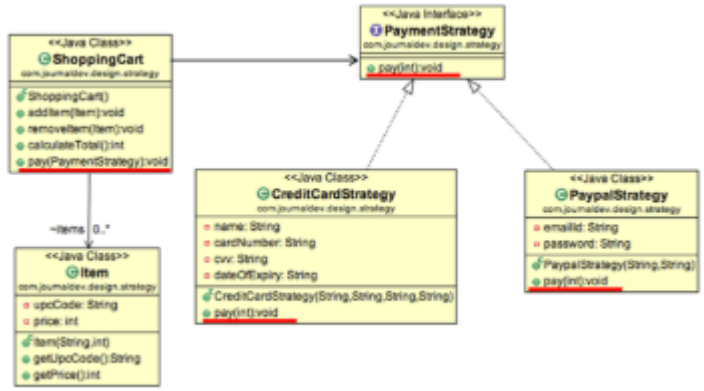
B：Adaptee 类用于将 client 的请求转化为对 Adapter 类的方法的调用
C：Adapter 类和 Adaptee 类之间的关系是 delegation
D：Adapter 和 ITarget 之间的关系是 inheritance
正确答案：C

2.



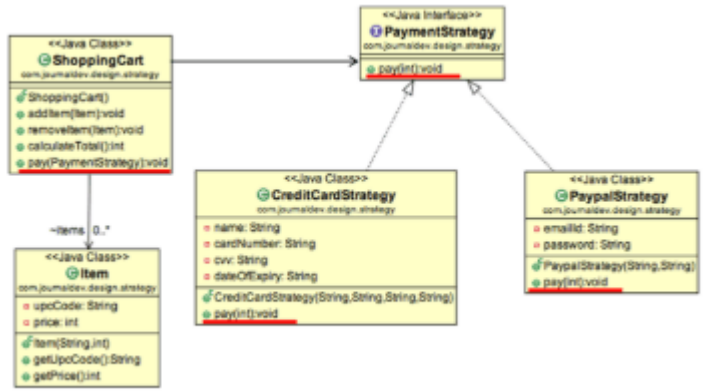
Adapter 模式カ何要设计
ITarget 接口，而不是由 client 直接靖求 Adapter 的 Request( )方法?
A:Client 对接口編程，无需了解 Adapter 炎
B:Adapter 炎可能友生变化,接口隔寓了变化，不会:影呵 client 代碣
C:Adaptee 炎可能友生变化，需要隔禽 client 与它的变化
D:Client 无法直接杓造 Adapter 的实例，故需要增加接口
正确答案：AB
3.
关于 Decorator 设计模式的说法，不正确的是
注:在以下选项中，A 代表装饰之前的类，B 代表装饰之后的类
A:A 和 B 二者具有共同的祖先类或实现共同的接口
B:B 中有一个成员变量,其类型为 A
C:B 中的某些方法中，通过
delegation 调用 A 的同名方法，并可扩展其他新功能
D:B 中不可以增加 A 中所没有的方法
正确答案 D
4.
Stack t = new SecureStack (
new SynchronizedStack (newUndoStack(s));
针对 Decorator 设计模式上述形式的 client 代码，以下说法不正确的是___
A:第三行的 s 的类型是 Stack 或其子类型
B:第三行的 s 的类型是 B UndoStack
C: 第一行的 t 的运行时类型是 SecureStack
D:第一行的 t 和第三行的 s，其实是指向内存中同一个对象的两个不同 alias
正确答案：BD
5.
针对 Facade 设计模式，以下说法不正确的是_
A:针对一个或多个已有的类，client 需要调用它们的多个方法，故该模式将这些调用统一封装为一 个方法对外提供
B:该模式降低了 client 端使用已有类的代价，减少了 client 与已有类之间的耦合度

C:该模式所构造的封装类中的方法一般用 static 类型
D:虽然减少了耦合度，该模式仍然需要 client 端显式构造出被封装类的实例
正确答案：D
6.



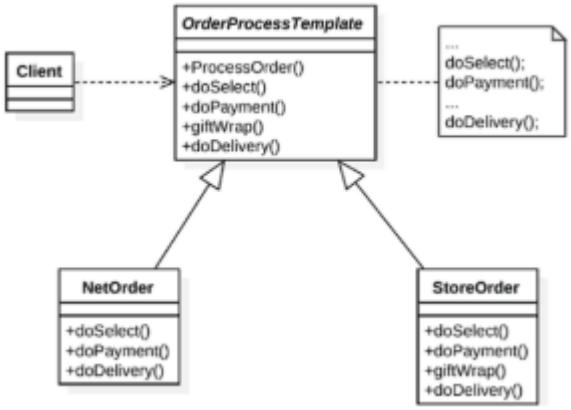
*．上图 Strategy 模式, 说法不正确的是_*
A：ShoppingCart 将 pay( )的职责 delegate 给了 PaymentStrategy
B: PaymentStrategy 是接口，client 的 pay 实际执行的是其某个实现类的 pay()Client 端不需要了解
C: PaymentStrategy 的任何实现类即可使用其 pay()
D: 若要扩展新的 pay( )策略，只需为 PaymentStrategy 增加新的实现类
正确答案：C

针对 C 选项：client 必须要知道自己希望 使用的是哪个 pay 策略，然后构造其对 象，传入 pay(PaymentStrategy p)方法， 才能调用 p 的 pay()方法。
7.



上图 Strategy 模式, 说法正确的是
ShoppingCart 将 pay( )的职责 delegate 给了 PaymentStrategy
PaymentStrategy 是接口，client 的 pay 实际执行的是其某个实现类的 pay()
若要扩展新的 pay( )策略，只需为 PaymentStrategy 增加新的实现类
不正确的
Client 端不需要了解 PaymentStrategy 的任何实现类即可使用其 pay()

client 必须要知道自己希望 使用的是哪个 pay 策略，然后构造其对 象，传入 pay(PaymentStrategy p)方法， 才能调用 p 的 pay()方法。
8.



上图显示的 Template 模式，不正确的是
上图中的 0rderProcessTemplate 是"模板”，可以是接口，也可以是抽象类
Client 可以更改对一系列 doXXX( )方法的调用次序
正确的
ProcessOrder()是模板方法，包含了对一系列操作的调用
如果 OrderProcessTemplate 中没有某个 doXXX( )方法的实现，则在 NetOrder 和 StoreOrder 中必须要实现它。
9.
关于 Iterator 模式，不正确的是
为了让你的类 A 具备 iteration 的能力，需要 A 实现 Iterator 接口
为了让你的类 A 具备 iteration 的能力，还需要构造一个实现 Iterable 接口的类作为符合 A 特定需求的迭代器
正确的
Java 里 Iterable 接口只有一个方法 iterator(),它返回一个迭代器对象
Java 里 Iterator 接口有三个方法 hasNext(),next()，remove( )

雨课堂知识点总结（十六）

5-2 组合与委派、框架复用
第一题
关于 delegation 的说法，正确的
某些功能不需要 ADT 自己来做，而是委托其他 ADT 来做(调用其他 ADT 的方法)
在类 1 的构造器里传入类 2 的对 B 象，即可在类 1 里使用传入的对象调用类 2 的操作
Delegation 发生在 object 层面而非 class 层面

不正确的
为实现类 1 对类 2 的 delegation,需要将类 2 作为类 1 的 rep 的一部分(即类 2 是类 1 的 field)
不必需。可以动态传入类 2 的对象,然后调用其方法即可。
```
class A {
void a(B b) {
b.add()
}
}
```
在上面的代码中，B 并未作为 A 的 rep 的一部分，只是在方法 a 中动态传入，然后调用 b 的 add()方法。
第二题
关于 Inheritance 和 delegation 的说法，正确的
如果类 1 需要使用类 2 的大部分乃至全部方法，则最好将类 1 设计为类 2 的子类
如果类 1 只需使用类 2 的小部分方法，则最好通过 delegation 实现对类 2 的方法调用
Delegation 发生在 object 层面，而 inheritance 发生在 class 层面
不正确的
用 A has a B 或 A usea B 表示 A 和 B 之间的 inheritance 关系，用 A isa B 表示_者之间的 delegation 关系
反了
第三题
关于 Composite over inheritance 原则，以下正确的是
若用继承，子类不得不继承父类的全部方法，但子类可能并不需要这么多方法
子类的不同对象可能具备不同的行为，若用继承实现，则不得不定义大量具有不同行为的子类，麻烦
若用继承，子类和父类之间的关系在编译阶段即硬性绑定，难以在运行时动态改变
若用 delegation,可隔离两个类之间的静态绑定关系，从而支持运行时实例之间关系的动态改变
第四题
若事物 o 既有 A 行为(例如"飞") ,也有 B 行为(例如“叫”),但具体如何实现这些行为可能会变化。那么 ready for change 的 Java 0OP 设计是
为 A 和 B 分别定义一个接口，各 B) 自分别定义自己的方法;为 o 设计类，将其具体行为 delegate 到 A 和 B 的具体实现类将两类行为分别放在不同接口里, o 的类通过 delegation 调用两类接口的具体实现。这是 CRP 原理的恰当应用。
不正确
定义一个接口，在其中为 A 和 B 分别定义- -组方法，然后为 o 设计一个类，实现该接口
如果 o 将来不再具备 A 或 B 的行为,那么需要修改该接口,会造成大量:依赖该接 C 的类随之修改。
为 o 设计类,在类中添加 A 和 B 的行为作为其一组方法
对接口编程,而非对类编程。

针对 A 设计一个类,针对 B 设计另一个类,然后让 o 的类同时继承 A 和 B 的类。
Java 不支持多重继承。
第五题
Lab2 的 P3 按照上述 UML 图进行设计,那么 GoGame 类的 place( )方法(落子)的 spec 应该为
void place(GoAction a, ...)
不正确
void place(Action a, ...)
void place(..)
void place()
.. 表示其他参数
第六题
Lab2 的 P3 按照上述 UML 图进行设计,这么设计的优点是
客户端只需了解 Game 接口，无需获知其他任何信息
可容易的扩展其他棋类游戏(只需为 Game 和 Action 两个接口添加新的实现类)
如果要扩展支持真实的围棋规则，只需改 GoAction 类即可，最小化改变范围
第七题
关于黑盒与白盒 framework 的说法，正确的
白盒框架里实现了若干方法,用户程序通过 override 这些方法即可改变框架的行为
在黑盒框架里，用户编写类来实现框架提供的接口，框架运行时动态调用用户写的类中的方法
白盒框架的基本原理是 inheritance,黑盒框架的基本原理是 delegation
不正确的
黑盒框架运行时的主程序是用户写的类，白盒框架的主程序是框架本身的类
第八题
上图中表示 delegation 关系的是
c 和 e
a 和 d，并非 ADT 之间的关系：这里的 “终端用户”就是使用软件的真人了。
第九题
上图中表示继承或实现关系的是
c 和 e
c 是用户开发的具体类实现了 Plugin 接 口，e 是用户定义的类继承了 framework 所提供的抽象类。

雨课堂知识点总结（十七）

8.4 动态性能分析方法与工具

1. 以下关于 Dynamic Program

Analysis 的说法，不正确的是__

A 根据程序执行的过程与结果，分析代码在时空性能方面所展现出的性质

B 对程序执行的性能没有影响

C 可用来发现程序中的“热点”语句，即哪些语句被频繁执行

D 只需执行次,即可比测试更容易发现程序中的性能弱点和bug

E 能够发现程序中各种不同类型的object分别占用了多少内存空间

正确答案：BD

2.关于 profiling 的三种策略，说法不正确的是

A 采用代码注入的策略，对程序性能的度量最为准确

B 采用 Instrumented VM 的策略，需针对不同的 VM 分别使用不同的 profiling 工具

C 采用 Sampling 的策略，对程序执行性能的影响最大

D 采用 Instrumented VM 和 Sampling 的策略，均不需对程序代码进行修改

E 采用代码注入的策略，不可能对目标代码进行动态注入

正确答案：ACE

3. 关于 Java 提供的若 Fprofiling 工具,说法不正确的是_

A 有工具可以获取 Java 程序运行时 JVM 管理的各 heap 区域的动态占用情况

B 有工具可以得知指定的 Java 程序所采用的 GC 策略

C 有工具可以对正在运行的 Java 程序的 JVM 内存配置进行参数的动态设置

D 即使不使用这些工具，当 Java 程序抛出 0utOfMemoryError 时, JVM 也能够自动导出内存溢出时刻的 heap dump

E 有工具可以获取当前时刻 Java 程序主线程的 call stack 的状态

正确答案：CD

4.

```
jstat -gcutil 21891 250 7
  S0     S1     E      O      M     CCS    YGC
  0.00  97.02  70.31  66.80  95.52  89.14      7
  0.00  97.02  86.23  66.80  95.52  89.14      7
  0.00  97.02  96.53  66.80  95.52  89.14      7
 91.03   0.00   1.98  68.19  95.89  91.24      8
 91.03   0.00  15.82  68.19  95.89  91.24      8
 91.03   0.00  17.80  68.19  95.89  91.24      8
 91.03   0.00  17.80  68.19  95.89  91.24      8
```

从该结果看，在第一次 GC 前后,survivor space 的占用比例增加了_ old generation 的占用比例增加了_

A 91.03%，1. 98%

B -5.99%，1.39%

C -94.55%，-97 .02%

D 91.03%，0.37%

正确答案：B

5.不能导出某个 Java 进程的 heap dump 文件的是_

A jmap

B jcmd

C jconsole

D jstack

E jhat

F Eclipse Memory Analyzer

正确答案：　DEF

6.使用 profiling 工具来监控你的 Lab3 程序时，发现 heap 中

出现了大量的 PhysicalObject 对象实例，占用了大量内存。可能的原因是___

A: 你的 Lab3 将 Physicalobject 设计 A 为 mutable,并对其采取了防御式拷贝策略以避免表示泄漏
B: 本次运行读入了一个大文件,故 B 构造 Circularorbi t 对象时不得不构造大量 Physicalobject 对象
C: 导出 heap 的时刻之前较长时间没有进行 GC,故大量无活性的 Physica lobject 仍处于内存中
D:你的 Lab3 对 Physicalobject 的生成(new)采用了"静态工厂”模式，导致 JVM 无法获取各
Physicalobject 对象的活性而无法及时 GC

正确答案：ABC

7.对代码进行 dynamic profiling,不需要在__时候进行

A: ADT 的初始版本完成后(包括完成了 Rep、方法、AF、RI、Spec、各方法的代码)

B:ADT 测试完成后(根据 spec 设计测试用例，用 JUnit 执行测试用例并获得结果后)

每次向 Git 进行一次 commit 之前

C: ADT 迭代开发结束，除性能之外的其他外部和内部质量指标的优化均已经达到期望

D: 交付用户之前，发现程序运行缓慢，与期望不符

正确答案：ABC

雨课堂知识点总结（十八）

10-1 Concurrency
第一题
以下是计算机系统中的 concurrency 现象?
A 手机上的一个 App 通过 5G 网络访问云端数据
B 四核 CPU,执行多道程序
C 使用 Observer 设计模式的 Java 程序，其中 Subject 类和 Observer 类的执行
D 一亿人同时登录 12306 网站抢购春运火车票
E 使用 JVM 参数-XX: +UseConcMarkSweepGC 启动的程序，运行时进行 GC
F 同一个 Java 程序内的两个线程，共享一个 mutable 的
答案：ABDEF

第二题
关于 process 和 thread 的说法,不正确的是
A 多个 process 之间不共享内存，而多个 thread 之间可共享内存
BCPU 的某个核,在特定时间点只能运行单一 process, 但可并行执行多个 thread
C 手机上的 App 通过 5G 网络访问云服务器的资源，手机上和服务器上运行的是不同的 thread 而非不同的 process
D 一个 process 可以包含多个 thread,一个 thread 只能在一个 process 里运行
答案：BC

第三题
关于如何创建 thread 的说法，不正确的是
A 从 Thread 类派生子类 A，创建线程时(new A()).start()
B 类 A 实现 Runnable 接口，并实现其 run()方法，创建线程时
(new Thread(new A())).run()
C
new Thread(new Runnable(){
public void run() {...};}) .start();
D
new Thread(new Runnable(){
public void run() {...};}).run();
答案：BD

第四题
关于 time slicing, interleaving 和 race condition 的说法，正确的是
ATime slicing 由 OS 决定，但程序员可在代码中进行若干有限度的控制
B 如果某程序执行结果的正确与否依赖于 time slicing,那么意味着程序执行中产生了 racecondition
C 程序 interleaving 执行的基本单元是 Java 代码行
D 同一个并发程序的多次执行中的 time slicing 可能完全不同，因此 bug 很难复现，将此类 bug 形象的称为 Bohrbugs
答案：AB

第五题
关于 Java Thread 的 sleep( )和 interrput(),不正确的是__
A 若线程的 run( )代码中包含 Thread.sleep(100),意味着该线程停止执行 100ms, CPU 交给其他线程/进程使用
B 线程 t1 中包含代码 t1. interrupt(),意味着执行完该语句后 t1 被终止，不会再获得 time slicing
C 线程 t1 在 sleep( )期间可捕获到其他线程发来的中断信号并抛出 InterruptedException 异常
D 线程若捕获了抛出 InterruptedException 异常，则自动终止执行
答案：BD

第六题
Thread t = new Thread(new Runnable(){
public void run(){
try{
print("a");
.*.
Thread. sleep(200);
}catch(InterruptedException e){
print("b");

print("c");};});
t.start();
t. interrupt();
某类的 main( )代码如上所示，执行后可能的输出结果为
ac
abc

第七题

如果希望让线程执行结束之后再执行其他线程，即让其他线程暂停，需要调用
t.join();
用于“检测当前线程是否收到其他线程发来的中断信号”的 Thread 静态方法是
Thread. isinterrupted()
在线程类的 run( )方法中有以下代码，如果希望收到中断信号后终止线程执行，则 TODO 位置的语句应该是 return
try{
Thread. sleep(1000);
} catch(InterruptedException e){
TODO;}

雨课堂知识点总结（十九）

8-5 面向性能的代码调优
1. 以下关于代码调优的说法，不正确/不恰当的是

A 代码行数越少，代码的执行性能倾向于更好

B 每写完一个 method 的代码,最好对其性能进行优化，确保时空复杂性优化

C 直到软件开发完全结束、所有其他质量指标均已满足期望,再进行代码调优

D 每次进行代码调优前，必须要使用 profiling 工具进行性能监控和度量

E 每次代码调优之后、修改代码提交 Git 仓库之前，都需要进行 regression testing
正确答案：AB
2.
美于 Singleton 没汁模式，鋭法不正碉/不恰当的是__

A 符合孩模式的 ADT，不血具有 constructor,以避免 client 創建多个突例

B 符合垓模式的 ADT，其 rep 中血具有一-个 static 的 field,其炎型是該 ADT 本身

C 符合孩模式的 ADT，竝具各一-个方法 getInstance(),在核方法内进行垓 ADT 的 new 操作

D State 没汁模式中，每个 State 的具体子炎，可遵守 Singleton 模式进行突现
正确答案：AC
A 选项：每个类都有 constructor，只不 过该模式下的 constructor 是 private 的，不能让 client 调用
C 选项：除非用 lazy load 的策略，否则 getInstance 方法中只需直接返回 rep 中 的单例对象
3.
使用 Flyweight 模式实现文本编辑器中对各个"字母' 1 对象的复用，以下说法不正确的是____
A "字母的内容(例如 a、 b)是内特性，"字母"在编辑器中展示的字体、字号、颜色是"外特性"
B 即使"字母"对象的字体/字号/颜色可变化，但"字母" ADT 仍应该是 immutable 的

C "字母" ADT 的 rep 中需包含其内容、颜色、字体、字号等属性

D 只需要 26 个大写字母和 26 个小写字母的对象实例，即足够 client 在编辑器中使用"字母"的功能

E 客户端代码中需要维护某些数据来管理或计算每个"字母"对象在文档中不同位置的字体、字号、颜色等。

正确答案：C

4.
关于 Prototype 模式，说法不正确的是

A 遵循该模式的 ADT，要实现 Cloneable 接口并 override0bject 的 clone( )方法

B 0bject 的 clone( )方法，缺省实现的是 shallow copy 而非 deep copy

C 若在调用 ADT 的 clone( )时抛出 CloneNotSupportedException,则意味着该 ADT 没有 override0bject 的 clone( )

D 在进行对象构造时，相比于直接使用构造器来说，使用 clone()的代价较小、构造时间更短
正确答案：CD
5.
关于 Object Pool 的说法，不正确的是_
A 不采用该模式时，如果一个对象实例不再有活性，即会被 GC;采用该模式时，将对象实例加入 pool,相当于强行保持其活性而不会被 GC
B 当 client 需要一个对象实例时,先到 pool 中获取，使用完之后，再归还回去

C 原本可被 GC 的对象，现在要留在 pool 中，导致内存浪费，但节省了频繁的 new 和 GC 的时间

D 不能把不同类型的对象实例放在同一个 pool 中进行管理，需要分别设定不同的 pool

正确答案：D

6.
关于 Singleton、Flyweight、
Object Pool 设计模式之间的异同，说法不正确的是___

A Singleton 相当于把每个 class 做成了 pool,其中包含唯一的对象实例，且是 static 的

B Flyweight 相当于只使用一个对象实例来表示一组具有相同内属性但不同外属性的对象

C Flyweight 模式维持一个 pool,pool 中包含一组具有不同内属性的对象实例

D 这三种模式的 client 使用某个对象实例时，均需要从 pool 中申请获得实例、用完归还给 pool

E 目的都是为了降低 new 对象实例时的时间代价，降低了程序运行时内存消耗
正确答案： DE
7.
以下
能够减少创建 object 的数降低 GC 的代价
A 方法中的临时变量尽可能使用
primitive 数据类型(e.g. double),减少使用其对象数据类型(e.g.,Double)
B 除非迫不得已(ADT 安全性要求),不要使用防御式拷贝

C 对频繁使用的对象用 object pool 进行 canonicalization,哪怕是类似于 Integer 这样的"小”对象

D 尽量使用 String a =“foo”的方式来定义常量字符串，避免 new String(“foo")

E 尽可能使用类的静态工厂方法进行对象创建，避免直接用 new

F 在为 ADT 设计 rep 的时候，除非迫不得已，最好用简单数据类型

正确答案：ABCDEF

雨课堂知识点总结（二十）

8-2 内存性能与垃圾回收
第一题
关于内存管理的三种模式(static、stack、heap) ，不正确的是
A Static 在编译阶段为各变量分配内存，不支持运行时变量扩展内存，但支持运行时为新变量分配内存
B Stack 因为结构简单，不支持存储结构复杂的变量(例如数组、对象等)
C 在 heap 中分配内存的变量,其内存大小在运行时可动态扩展或收缩
D Static 和 stack 两种模式都不支持对变量所占内存的动态回收
E Java 中 heap 用于复杂结构的数组或对象等变量的内存分配，而简单数据类

型的变量在 stack 中分配内存
答案：ADE

第二题
类 A 有一个静态成员变量 int a 和一个非静态成员变量 Date b,它的某个方法内使用了一个局部变量 Calendar C。JVM 在为 a、b、c 三个变量分配内存时，分别在____中分配
A Stack / Stack / Heap
B Stack / Method Area / Stack
C Heap / Stack / Stack
D Method Area / Heap/ Heap
答案：D

第三题
关于 GC 的说法，不正确的是
A GC 根据对象的“活性”(从 root 的可达性)来决定是否回收该对象的内存
B 如果某个对象内部不再有指向其他对象的 reference,则该对象一定是 dead,会被 GC
C 在 Java 中，GC 完全依赖于 JVM,程序员无法通过代码来主动改变某个对象的”活性”
D Defensive copy 策略大大增加了 GC 的负担，相当于用性能换取安全性
E 程序员在代码中人工实现 GC,若不恰当，可能导致内存泄漏
答案：BC

第四题
Java 中针对 heap 的各种 GC 策略，以下说法不正确的是
A Mark-sweep 检查内存中各对象是 live 还是 dead,对 dead 对象做出标记，进而将其清理
B 每次进行 Mark-compact 策略的 GC 之后，heap 中可被再分配的区域是连续的
C Copy 是四种策略中唯一不需要检查对象 live/dead 的 GC 策略
D Reference counting 相对于其他三种 GC 策略，内存中 dead 对象的 zombie time 最短
答案：C

第五题
针对 JVM 内存管理和 GC 的说法，不正确的是
A 相对于 old generation space 中对象的存活时间，young generation space 中对象的存活时间更短
B 对存活时间非常长的对象，经历 B) 多次 GC 后，在 heap 中位置的迁移路线是 eden>(S0 或 S1)>(S1 或 S0)>... >old generation space
C Old generation space 的 GC 是 full GC,当 young generationspace 满了的时候触发执行
D 若针对 young generation space 的 minor GC 的发生频度很高，则需扩大该

区域的内存大小
答案：C

第六题
- Xms512m- Xmx1024m
- XX : Metaspacesize=128m
- XX : MaxMetaspaceSi ze=192m- XX: NewSize=128m
- XX : MaxNewSize= 256m- XX: SurvivorRatio=8
- XX : MaxHeapFreeRatio=90
按上述参数配置，Old Generation 和(S0+S1)的最大值分别是
A 1024m, 25.6m
B 768m, 51.2m
C 896m, 51.2m
D 576m, 25.6m
答案：C

第七题
- Xms 512m- Xmx1024m
- XX :MetaspaceSize=128m
- XX : MaxMetaspaceSize=192m-XX: NewSize=128m
- XX :MaxNewSize= 256m-XX:SurvivorRatio=8
-XX: MinHeapFreeRatio=10
按上述参数配置，若当前 young 和 oldgeneration 的尺寸分别为 1 28m 和 384m,在以下的时刻， JVM 会自动增加 young generation 的尺寸?
A S0+S1 的总占用超过 96m
B Eden 的总占用超过 56m
C Old generation 的占用超过 350m
D S0+S1+ Eden 的总占用超过 120m
答案：D

第八题
- XX: +DisableExplicitGC-XX:+PrintGCDetails
- XX: +Us eConcMarkSweepGC 一 XX:ParallelCMSThreads=12-verbose:gc
- XX : +HeapDumpOnOutofMemoryError-XX:+PrintGCTimeStamps
-Xloggc:../ ../logs/gc-console.log
按以上参数配置，以下不正确的是
A 程序运行时可将 GC 过程计入日志文件，并记录历次 GC 时间戳
B 使用串行的 GC 策略进行 GC,GC 时会短暂停止程序执行
C 开发者可手工在程序代码中使用 System. gc()以提升 GC 性能
D 使用 12 个 GC 线程，降低了 GC 对程序执行性能的影响
答案：BC