华中科技大学计算机科学与技术学院 2019-2020 学年第 2 学期考试试卷

Java 语言程序设计

一、第一大题

第一小题

1、(1) 不正确,super()函数要放在第一句才行,应该先构造父类

(2) 正确,因为 super 是第一句,并可可以访问 myInterestRate,并且让这个成员为 0

(3) 不正确,因为 super 是第一句,但应该要初始化所有成员,比如说 myInterestRate

2、(1) 不正确,因为我们不能访问父类的私有成员

(2) getbalance 的返回值为右值,不能作为左值

(3) 不正确,因为这样的 balance 不能赋值

(4) 正确,这样子可以完美地进行赋值

3、(1)正确

(2)因为 balance 没有初始化,所以说错误

(3)因为 balance 没有初始化,所以说错误

4、(1) 不能访问父类私有成员 myBalance

(2) 不正确,这样会访问被覆盖的父类成员

(3) 正确,这样会访问父类有的方法成员

(4) 不能访问父类的私有成员 myBalance

5、(1)不正确,因为 BankAccount 类型变量没有 addInterst 方法

(2) 正确,s 是 SavingsAccount 类型变量,并且父类也有 withDraw 方法

(3) 正确

第二小题

(1)首先是早期绑定:绑定到 method1()类型变量,输出 A's method1

(2)首先是早期绑定,绑定到 method1()类型变量,接着晚期绑定,绑定到 B 内部的变量,输出 B 's method1

(3)首先是早期绑定:绑定到 method1(int)类型变量,输出 A's static method1

(4)首先是早期绑定:绑定到 method1(int)类型变量,输出 A's static method1,因为静态函数没有多态性

(5)这里会编译报错,因为 o1 的声明类型为 A,在函数里面没有匹配的重载

(6)这里会编译报错,因为 o2 的声明类型为 A,在函数里面没有匹配的重载

(7)首先是早期绑定,绑定到 B 内部的 method2(),但是晚期绑定绑定失败,会报运行时异常

(8)首先是早期绑定,绑定到 method1()类型变量,接着晚期绑定,绑定到 B 内部的变量,输出 B 's method2

首先是早期绑定(重载):先在声明类内部寻找有没有匹配的重载,接着是晚期绑定,从本类往父类(沿继承链)寻找有没有匹配的晚期绑定(因为有继承性质,本质上是寻找本类里面有没有匹配的方法)

如果声明类型是子类,但是引用的对象是父类的话,这个时候去调用父类没有的方法(子类新定义的方法),会异常

如果声明类型里面没有与该类匹配的元素,就报错

二、第二大题

第一小题

```
(1)public BirthDay(int year,int month,int day){
    this.year=year;
    this.month=month;
    this.day=day;
}
(2)public boolean equals(Object obj){
    if(obj instanceof BirthDay){
        Birthday newObj = (BirthDay)obj;
    }
    else{
        return false;
    }
    return(this.year==newObj.year)&&(this.month==newObj.month)&&(this.day==newObj.day)
;
}
(3)public String toStirng(){
    return this.year+".”+this.month+".”+this.day;
}
(4)public String compareTo(BirthDay o){
    if(o.year==this.year){
        if(o.month==this.month){
            if(o.day==this.day) return 0;
            else return this.day>o.day?1:-1;
        }
        else{
            return this.month>o.month?1:-1;
        }
    }
    else{
        return this.year>o.year?1:-1;
    }
```

第二大题

```
(1)public Person(String id,int year,int month,int day){
    this.BirthDay(year,month,day);
    this.id=id;
}
(2)public equals(Object obj){
    if(obj instanceof Person){
        Person newObj = (Person)obj;
    }
    else{
        return false;
```

```
    }
    return this.id.equals(newObj.id)&&this.birthday.equals(newObj.birthday);
}
(3)public toString(){
    return this.brithday.toString()+"\nId: "+this.id;
}
(4)public int compareTo(Person o){
    return this.birthday.compareTo(o.birthday);
}
```

第三小问

```
(1)new List<Person>( );
(2)for(Person p:list){
    System.println(p.toString());
}
```

(3)元素的 compareTo 方法,结果:

1999.5.6 id: 2
2000.1.25 id: 1
2002.7.22 id: 3

(4)不同在:会出现两次 2000.1.25 id=1 的人,因为 Object 里默认的 equals 仅仅是比较元素的内存空间是不是相同而已

三、第三大题

第一小问

```
public static int reverseInt(int n){
    String s=Integer.toString(i);
    s=s.reverse();
    return Integer.parsenInt(s);
}
```

第二小问

```
(1)public class StringParser{
    public static List<String>(String lineString){
        String[] strList = lineString.split(" ");
        List<String> newList = new List<>(strList);
        return newList;
    }
}
(2)public class NonAlphabetWordFilter implement Filter{
    boolean accept(String s){
        for(char ch:s){
            if(ch<=40H||(ch>'Z'&&ch<'a')||ch>'z'){
                return false;
            }
        }
    }
    return true;
```

```
}
(3)public class StringParser{
    public static List<String>(String lineString,Filer filter){
        String[] strList = lineString.split(“ ”);
        List<String> newList = new List<>();
        for(String str:strList){
            if(filter.accept){
                newList.append(str);
            }
        }
        return newList;
    }
}
(4)public static void main(String[] args){
    String line = “Hello java and python 123 C++”;
    Filter fil = new NonAlphabet();
    StringParse sParse = new StringParse();
    List<String> newList=sParse.(line,fil);
}
```

四、第四大题

1、public class ComputableComparableTuple<T1 extends Computable&Comparable,T2 extends Computable&Comparable>implement Computable,Comparable

```
2、public ComputableComparableTuple<T1,T2> add(ComputableComparale<T1,T2>y){
    this.first=this.first.add(y.first);
    this.second=this.second.add(y.second);
    return this;
}
3、public ComputableComparable<T1,T2> substract(ComputableComparale<T1,T2>y){
    this.first=this.first.substract(y.first);
    this.second=this.second.substract(y.second);
    return this;
}
4、public int compareTo(ComputableComparale<T1,T2>o){
    if(this.first.compareTo(o.first)!=0){
        return this.first.compareTo(o.first);
    }
    else return this.second.compareTo(o.second);
}
5、class IntComputable implement Comparable<IntComputable>{
    private int value = 0;
    IntComputable add(IntComputable y){
        IntComputable newObj = new IntComputable();
        newObj.value=y.value+value;
        return newObj;
```

```
    }
    IntComputable substract(IntComputable y){
        IntComputable newObj = new IntComputable();
        newObj.value=value-y.value;
        return newObj;
    }

}
```

五、第五大题

第一小题

```
class FullPriceItem extends Item{
    protected double price;
    public FullPriceItem(String name,double price){
        super(name);
        this.price=price;
    }
    public Object clone throws CloneNotSupportedException{
        FullPriceItem newObj = (FullPriceItem)super.clone();
        newObj.price=this.price;
        return newObj;
    }
}
class DiscountItem extends FullPriceItem{
    public DiscountItem(String name,double price,double discount){
        super(name,price);
        this.discount=discount;
    }
    public double salePrice(){
        return price*(1-discount);
    }
    public Object clone() throws CloneNotSupportedException{
        DiscountItem newObj = (DiscountItem)super.clone();
        newObj.discount=this.discountl
        return newObj;
    }
}
```

第二小问

```
class WareHouse implements itemIterator{
    private int position1 = 0;
    private int position2 = 0;
    public WareHouse(List<Item>fullPriceItems,List<Item> discountPriceItems){
        this.fullPriceItems=fullPriceItems;
```

```
            this.discountPriceItems=discountPriceItems;
        }
        public Item next(){
            if(position1<fullPriceItems.size()){
                return fullPriceItems.get(position1++);
            }
            else if(position2<discountPriceItems.size()){
                return discountPriceItems.get(position2++);
            }
            else return null;
        }
        public boolean hasnext(){
            return position1<fullPriceItems.size()||position2<discountPriceItems.size();
        }
        public ItemIterator iterator(){
            return this;
        }
}
```

第三小问

```
public static double totalPrice(ItemIterator it){
        double res = 0;
        if(it.hasnext()){
            res+=it.next().salePrice();
        }
        return res;
}
```