Principles of Java Language with Applications, PIC 20A
Omri Azencot
Winter 2019

UCLA

Midterm Exam
Monday, February 11, 2019
50 minutes, 7 questions, 100 points, 7 pages

While we don't expect you will need more space than provided,
you may continue on the back of the pages.
This exam is closed book and closed notes.

Student name: [redacted]

Teaching staff signature: _____

85/100

Do not turn to the next page until the start of the exam.

1. (4 points) Consider the code in files `A.java` and `B.java`. Both files are in the same folder `tmp/`. → package-private?

```
// file A.java
public class A {
  double num;
}
```

default setting modifier is "private" ← → this is private to A only. private values aren't inherited

```
// file B.java
public class B extends A {
  public B() { num = 1.0; }
}
```

What can be said about `B`'s constructor?

~~A. It uses an undefined variable num~~ → technically "num" is defined, just in A class only

B. It initializes the num field of `class A` → this is allowed if package-private → by rules of inheritance, privates not inherited

C. It performs an incorrect access to the field num which is not inherited

~~D. It incorrectly calls the constructor of `class A`~~ → this is just not true. No calls to A's constructor aren't even made.

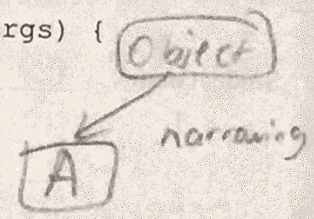
I'm going with rules of private inheritance

2. (4 points) Consider the following code

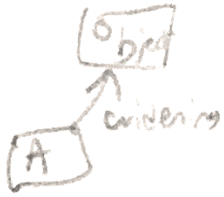
```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    Object o = (Object) a;
    Object o2 = new Object();
    A a2 = (A) o2;
  }
}
```

Ok; widening

Not ok; Narrowing



12/20



A. The widening conversion is correct, the narrowing conversion will cause a runtime error

B. Both conversions are correct

C. Both conversions are not required for the code to compile and work

D. None of the above answers

3. (4 points) What is the purpose of `@Override` annotations?

A. They are mostly for debugging purposes and not needed in production

B. They make the code longer and more complex

C. They augment the code, facilitating better readability and allowing to spot errors

D. Answers A and B

4. (4 points) Consider the following code

```
class A {
  public static String func() { return "A"; }
}

class B extends A {
  public static String func() { return "B"; }
}

public class Test {
  public static main(String[] args) {
    A a = new B();
    System.out.println( a.func() );
  }
}
```

What should be the output?

A. BA → impossible
B. AB
C. B
D. A

B extends A & we widen convert in main() B to A. Because of polymorphism, it will call B's version of func(): "B"

5. (4 points) How can you force users to not instantiate your class?

A. Make all c'tors `private` → this will make all possible ways of creating class outside of it not possible, can't use constructors
B. Make your class `abstract` → will prevent any instantiations of it & can only make classes that extend the abstract class
C. Do not provide any c'tors → does not work because no constructors means we can still call default constructor to instantiate the class, if class is public.
D. A+B
E. B+C
F. A+C

6. (40 points) Write the `class Array` which supports various operations between arrays. Specifically, your code will implement the following features
   - `Concatenate`: given two `Arrays`, concatenate them one after the other
   - `Intersect`: given two `Arrays`, return an `Array` storing their intersection

   The following code example demonstrates how `Array` might be used

```
public class Test {
  public static void main(String[] args) {
    Complex[] carr1 = new Complex[]
            { new Complex(1,2), new Complex(3,4) };

    Complex[] carr2 = new Complex[]
            { new Complex(3,4), new Complex(5,6) };

    Array arr1 = new Array( carr1 );
    Array arr2 = new Array( carr2 );

    Array arr3 = arr1.Concatenate( arr2 );
    Array arr4 = arr1.Intersect( arr2 );

    arr3.print();
    arr4.print();
  }
}
```

Object.equals

After executing the above `main` method, `arr3` holds all of the four `Complex` objects, and thus the output of `arr3.print()` is `1+i2, 3+i4, 3+i4, 5+i6`. The second array, `arr4`, stores only one object, namely `arr4.print()` outputs `3+i4`.

Notes and remarks:

- While you are required to implement `Concatenate` and `Intersect`, these methods alone are not enough. In particular, you might be needed to implement additional features for the class to work properly with respect to the example code. For instance, you need to implement a `print()` method for the `class Array`.
- Your code should support only arrays of reference types.
- Use `equals()` to determine if two reference objects are equal

35/40

```
public class Array {
    public Object[] arr;          // private  (-1)
    public Array(Object[] arr) {
        this.arr = arr;           dep copy → init arr → ?
    }
    public void print() {
        for (int i=0; i< arr.length-1; i++){
            System.out.print(arr[i] + ", ");
        }
        System.out.println(arr[arr.length - 1]);
    }
    public Array Concatenate(Array other) {
        Object[] concat = new Object[this.arr.length + other.arr.length];
        for (int i=0; i< this.arr.length; i++){
            concat[i] = this.arr[i];
        }
        int n = this.arr.length;
        for (int i=0; i< other.arr.length; i++) {
            concat[i+n] = other.arr[i];
        }
        return new Array(concat);
    }
    public Array Intersect(Array other) {
        ArrayList<Object> inter = new ArrayList<Object>();
        for (int i = 0; i< this.arr.length; i++) {
            Object obj= this.arr[i];
            for (int j=0; j< other.arr.length; j++) {
                Object obj2= other.arr[j];
                if obj.equals(obj2) {
                    inter.add(obj);
                    break; // prevent duplicates in intersection
                }
            }
        }
        Object[] interArray= inter.toArray(); // arraylist method to get array version of it. Might be "get Array()" I forgot.
        return new Array(interArray);
    }
}
```

7. (40 points) Write a Monte Carlo algorithm in Java that computes the probability to win in a game of craps. In this game, we roll two dice and find their sum. There are three possible scenarios leading to a win or loss:
   - If the sum is 7 or 11, we win
   - If the sum is 2, 3, or 12, we lose
   - Otherwise, we roll the dice until we get the initial sum (win) or a sum of 7 (lose)

   For instance, the following sequences result in a win:
   - (1,6) // 7
   - (6,5) // 11
   - (2,3), (1,1), (1,4) // initial sum 5

   On the other hand, the following sequences result in a loss:
   - (1,1) // 2
   - (2,1) // 3
   - (4,4), (1,2), (1,6) // 7

   Tips and notes:
   - Write a method that plays a single game of craps
   - Once you have it, you can use it in a loop to count wins vs. #games.
   - The class `Random` provides `nextInt(int n)` method which returns a random integer between 0 to n-1

38/40

```
// assume class declared in headers        import Random ~/
public boolean playCraps(){ // returns true if win, else false
    Random rand = new Random();
    int roll1 = rand.nextInt(6)+1; // 1-6 inclusive
    int roll2 = rand.nextInt(6)+1;
    int initSum = roll1+roll2;
    switch(initSum){
        case 7:
        case 11:
            return true;
        case 2:
        case 3:
        case 12:
            return false;
    }
    int sum=0;
    do{
        roll1 = rand.nextInt(6)+1;
        roll2 = rand.nextInt(6)+1;
        sum = roll1+roll2;
    } while (sum!=initSum && sum!=7);
    if(sum==initSum){
        return true;
    }else{
        return false; // Sum is 7
    }
}
public double monteCarlo(int N){ //N is # of trials
    double sum=0; // # wins we set playing game
    for(int i=0; i<N; i++){
        if(playCraps()){
            sum=sum+1;
        }
    }
    return sum/N;
}          cast to double
                ~/
```