CS 61B Fall 2020

Inheritance

Exam Prep Discussion 4: September 14, 2020

1 Athletes

Suppose we have the `Person`, `Athlete`, and `SoccerPlayer` classes defined below.

```
1  class Person {
2      void watch(Athlete other) { System.out.println("wow"); }
3      void speakTo(Athlete other) { System.out.println("kudos"); }
4  }
5
6  class Athlete extends Person {
7      void speakTo(Person other) { System.out.println("take notes"); }
8      void play(Athlete other) { System.out.println("game on"); }
9  }
10
11 class SoccerPlayer extends Athlete {
12     void speakTo(SoccerPlayer other) { System.out.println("howdy"); }
13     void speakTo(Person other) { System.out.println("hmph"); }
14 }
```

(a) For each line below, write what, if anything, is printed after its execution. Write CE if there is a compiler error and RE if there is a runtime error. If a line errors, continue executing the rest of the lines.

```
1  Person itai = new Person();
2
3  SoccerPlayer shivani = new Person();
4
5  Athlete sohum = new SoccerPlayer();
6
7  Person jack = new Athlete();
8
9  Athlete anjali = new Athlete();
10
11 SoccerPlayer chirasree = new SoccerPlayer();
12
13 itai.watch(chirasree);
14
15 chirasree.watch(itai);
16
17 jack.speakTo(anjali);
18
19 anjali.speakTo(sohum); // this one is really tricky!
20
```

```
sohum.speakTo(jack);

jack.play(chirasree);

((SoccerPlayer) jack).speakTo(chirasree);

((SoccerPlayer) sohum).speakTo(itai);

chirasree.speakTo((SoccerPlayer) sohum);

((Person) chirasree).speakTo(itai);
```

```
Person itai = new Person();
SoccerPlayer shivani = new Person(); // CE
Athlete sohum = new SoccerPlayer();
Person jack = new Athlete();
Athlete anjali = new Athlete();
SoccerPlayer chirasree = new SoccerPlayer();
itai.watch(chirasree); // wow
chirasree.watch(itai); // CE
jack.speakTo(anjali); // kudos
anjali.speakTo(sohum); // kudos
sohum.speakTo(jack) // hmph
jack.play(chirasree); // CE
((SoccerPlayer) jack).speakTo(chirasree); // RE
((SoccerPlayer) sohum).speakTo(itai); // hmph
chirasree.speakTo((SoccerPlayer) sohum); // howdy
((Person) chirasree).speakTo(itai); // CE
```

(b) Some of the errors above can be resolved by **adding/removing casting**. For this part, resolve those errors, and write the modified function calls below. If multiple casts would suffice, choose the most permissive one. Note: you cannot resolve a compile error by creating a runtime error!

```
((Athlete) jack).play(chirasree);
jack.speakTo(chirasree);
chirasree.speakTo(itai);
```

2 Dynamic Method Selection

Modify the code below so that the max method of DMSList works properly. Assume all numbers inserted into DMSList are positive, and we only insert using `insertFront`. You may not change anything in the given code. You may only fill in blanks. You may not need all blanks. (Spring '16, MT1)

```
1  public class DMSList {
2      private IntNode sentinel;
3      public DMSList() {
4          sentinel = new IntNode(-1000, ____________________);
5      }
6      public class IntNode {
7          public int item;
8          public IntNode next;
9          public IntNode(int i, IntNode h) {
10             item = i;
11             next = h;
12         }
13         public int max() {
14             return Math.max(item, next.max());
15         }
16     }
17     public ___________________________________ {
18
19     ____________________________________________________________
20
21     ____________________________________________________________
22
23     ____________________________________________________________
24
25     ____________________________________________________________
26
27     ____________________________________________________________
28
29     ____________________________________________________________
30
31     ____________________________________________________________
32
33     ____________________________________________________________
34     }
35     /* Returns 0 if list is empty. Otherwise, returns the max element. */
36     public int max() {
37         return sentinel.next.max();
38     }
39     public void insertFront(int x) { sentinel.next = new IntNode(x, sentinel.next); }
40 }
```

Solution:

```
public class DMSList {
    private IntNode sentinel;
    public DMSList() {
        sentinel = new IntNode(-1000, new LastIntNode());
    }
    public class IntNode {
        public int item;
        public IntNode next;
        public IntNode(int i, IntNode h) {
            item = i;
            next = h;
        }
        public int max() {
            return Math.max(item, next.max());
        }
    }
    public class LastIntNode extends IntNode {
        public LastIntNode() {
            super(0, null);
        }
        @Override
        public int max() {
            return 0;
        }
    }
    /* Returns 0 if list is empty. Otherwise, returns the max element. */
    public int max() {
        return sentinel.next.max();
    }
    public void insertFront(int x) {
        sentinel.next = new IntNode(x, sentinel.next);
    }
}
```

3 Challenge: A Puzzle

Consider the **partially** filled classes for `A` and `B` as defined below:

```
1  public class A {
2      public static void main(String[] args) {
3          ___ y = new ___();
4          ___ z = new ___();
5      }
6
7      int fish(A other) {
8          return 1;
9      }
10
11     int fish(B other) {
12         return 2;
13     }
14 }
15
16 class B extends A {
17     @Override
18     int fish(B other) {
19         return 3;
20     }
21 }
```

Note that the only missing pieces of the classes above are static/dynamic types! Fill in the **five** blanks with the appropriate static/dynamic type — `A` or `B` — such that the following are true:

1. `y.fish(z)` equals `z.fish(z)`
2. `z.fish(y)` equals `y.fish(y)`
3. `z.fish(z)` does not equal `y.fish(y)`

Solution:

```
1  public class A {
2      public static void main(String[] args) {
3          A y = new B();
4          B z = new B();
5      }
6      ...
7  }
```