CS 61B Spring 2021 — Inheritance

Exam Prep Discussion 4: February 8, 2021

1 Athletes

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

```
1  class Person {
2      void speakTo(Person other) { System.out.println("kudos"); }
3      void watch(SoccerPlayer other) { System.out.println("wow"); }
4  }
5
6  class Athlete extends Person {
7      void speakTo(Athlete other) { System.out.println("take notes"); }
8      void watch(Athlete other) { System.out.println("game on"); }
9  }
10
11 class SoccerPlayer extends Athlete {
12     void speakTo(Athlete other) { System.out.println("respect"); }
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      CE
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       wow
15 jack.watch(sohum);
16       CE
17 itai.speakTo(sohum);
18       kudos
19 jack.speakTo(anjali);
20       kudos
```

firstly, look in the method, then compare it with the method with the same signature from the subclass.

```
21 anjali.speakTo(chirasree);
22
23 sohum.speakTo(itai);
24
25 chirasree.speakTo((SoccerPlayer) sohum);
26
27 sohum.watch(itai);
28
29 sohum.watch((Athlete) itai);
30
31 ((Athlete) jack).speakTo(anjali);
32
33 ((SoccerPlayer) jack).speakTo(chirasree);
34
35 ((Person) chirasree).speakTo(itai);
```

take notes
hmph
respect
CE
RE
take notes
RE
hmph

(b) You may have noticed that `jack.watch(sohum)` produces a compile error. Interestingly, we can resolve this error by **adding casting**! List two fixes that would resolve this error. The first fix should print `wow`. The second fix should print `game on`. Each fix may cast either `jack` or `sohum`.

1. jack.watch((SoccerPlayer) sohum)
2. ((Athlete)jack).watch(sohum)

(c) Now let's try resolving as many of the remaining errors from above by **adding or removing casting**! For each error that can be resolved with casting, write the modified function call below. Note that you cannot resolve a compile error by creating a runtime error! Also note that not all, or any, of the errors may be resolved.

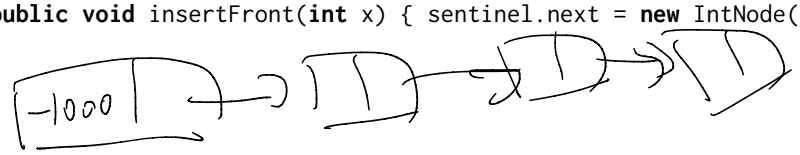
15. ((Athlete)jack).watch(sohum)
jack.watch((SoccerPlayer) sohum)

33. jack.speakTo(chirasree)

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, ____new LastIntNode()____);
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 ____class LastIntNode extends IntNode____ {
18
19     ____public LastIntNode() {____
20
21     ____super(0, null);____
22
23     ____}____
24
25     ____@override____
26
27     ____public int max(){____
28
29     ____return 0;____
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 }
```



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          _A_ y = new _B_();
4          _B_ z = new _B_();
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 **four** 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)`