CS 61B Spring 2018 — Inheritance and Testing

Exam Prep 4: February 11, 2019

1 Playing with Puppers

Suppose we have the Dog and Corgi classes which are a defined below with a few methods but no implementation shown. (modified from Spring '16, MT1)

```
1  public class Dog {
2      public Dog(){ /* D1 */ }
3      public void bark(Dog d) { /* Method A */ }
4  }
5
6  public class Corgi extends Dog {
7      public Corgi(){ /* C1 */ }
8      public void bark(Corgi c) { /* Method B */ }
9      @Override
10     public void bark(Dog d) { /* Method C */ }
11     public void play(Dog d) { /* Method D */ }
12     public void play(Corgi c) { /* Method E */ }
13 }
```

For the following main method, at each call to play or bark, tell us what happens at **runtime** by selecting which method is run or if there is a compiler error or runtime error. If you have a compile time error, you cannot run your code, and thus cannot have a runtime error

```
1  public static void main(String[] args) {
2      Corgi c = new Corgi();         Compile-Error   Runtime-Error  C1  D1
3      Dog d = new Corgi();           Compile-Error   Runtime-Error  C1  D1
4      //There is always an implicit call to the superclass's constructor.
5      Dog d2 = new Dog();            Compile-Error   Runtime-Error  C1  D1
6      Corgi c2 = new Dog();          Compile-Error   Runtime-Error  C1  D1
7      Corgi c3 = (Corgi) new Dog(); Compile-Error   Runtime-Error  C1  D1
8      //During compile time, we can cast an object along a class's heirarchy with no
9      //problem. At runtime, java is upset that the Dog instance "is not" a Corgi. That
10     //is, a Dog does not extend from Corgi. However, the dog is instantiated before
11     //java attempts to assign it.
12
13     d.play(d);         Compile-Error   Runtime-Error   A   B   C   D   E
14     d.play(c);         Compile-Error   Runtime-Error   A   B   C   D   E
15     //d's static type Dog does not have a play method.
16
17     c.play(d);         Compile-Error   Runtime-Error   A   B   C   D   E
```

```
18      //At compile time, we check c's static type, Corgi, does have a play method that
19      //takes in a Dog. At runtime, we look at c's dynamic type, Corgi, for a play method.
20      //Here we see play is overloaded, so we pick the method with the "more specific"
21      //parameters relative to our arguments, which is method D.
22
23      c.play(c);        Compile-Error   Runtime-Error   A   B   C   D   E
24      //Same as previous.
25
26      c.bark(d);        Compile-Error   Runtime-Error   A   B   C   D   E
27      c.bark(c);        Compile-Error   Runtime-Error   A   B   C   D   E
28      d.bark(d);        Compile-Error   Runtime-Error   A   B   C   D   E
29      //We notice that bark is overloaded and overriden. As a reminder, dynamic method
30      //selection applies to overriden methods. Method C overrides Method A, and method B
31      //overloads C. For c.bark(c), the compiler had bound caller c's static type's bark to
32      //argument c's most specific static type, Corgi, thus binding method B.
33
34      d.bark((int) c);    Compile-Error   Runtime-Error  A   B   C   D   E
35      //During compile time, the compiler will complain that a Corgi "is not" an int.
36      //You can only cast up or down the heirarchy.
37
38      c.bark((Corgi) d2); Compile-Error   Runtime-Error  A   B   C   D   E
39      //During compile time, we check c's static type, Corgi,
40      //for a bark method that takes in a Corgi, which exists, so there is
41      // no compile time error. At runtime, java is upset that d2
42      // "is not" a Corgi. Note that the cast only temporarily
43      //changes the static type for this SPECIFIC line.
44  }
```

We encourage you to try inheritance problems here: link. Please post on piazza if you have questions!

General flow for one argument methods, suppose we have `a.call(b)`: [ST = Static type, DT = dynamic type].

1. During compile time, java only cares about static types. First, check if a's ST, or its superclasses, has a method that takes in the ST of b.
   (a) If not, check a's superclasses for a method that takes in ST of b.
   (b) If not, check if any of the methods take in supertype of ST of b, as we are looking for b's "is-a" relationships. Start from a's ST methods and move up from its superclass.
   (c) If still not, Compile-Error!
2. Take a snapshot of the method found.
   (a) The method **signature** that is choosen at runtime will try to exactly match with our snapshot. The signature consists of the method name, and the number and type of its paramaters.

3. During runtime, if `call` is an overriden method, then run a's dynamic type's `call` method. If `call` is an overloaded method, then run the most specific snapshot.

4. Runtime errors can consist of downcasting (as seen in `Corgi c3 = (Corgi) new Dog();`), but also many that are not related to inheritance (NullPointerException, IndexOutOfBoundException, etc).

Notes:

- If a method is overloaded and overriden, as bark is above, the compiler will bind the method first.

- Dynamic method selection has no interaction with assignment.

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 between 'sentinel' and 'sentinel.next'. You may not change anything in the given code. You may only fill in blanks. You may not need all blanks. (Spring '17, MT1)

```
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();
    }
}
```

3 SList Debugging and Testing

Consider the SList, a linked list with a sentinel, implementation below. (Spring '16 MT1)

```
1  public class SList {
2    public class IntNode {
3      public int item;
4      public IntNode next;
5      public IntNode(int i, IntNode n) {
6        item = i;
7        next = n;
8      }
9    }
10   private static IntNode sentinel; // static sentinel is the flaw
11
12   public SList() {
13     sentinel = new IntNode(982734, null);
14   }
15   public void insertFront(int x) {
16     sentinel.next = new IntNode(x, sentinel.next);
17   }
18   public int getFront() {
19     if (sentinel.next == null) {
20       return -1;
21     }
22     return sentinel.next.item;
23   }
24 }
```

Write a JUnit test that fails on the code above, but would pass on a correct implementation. You may use any JUnit methods like assertEquals, assertNotEquals, assertTrue, assertFalse, etc. Hint: Create at least two instances.

```
1  @Test
2  public void test() {
3    SList s1 = new SList();
4    SList s2 = new SList();
5    s1.insertFront(1);
6    s2.insertFront(2);
7    assertNotEquals(s1.getFront(), s2.getFront());
8    assertEquals(1, s1.getFront()); /* also fails */
9  }
```

Click to open in java visualizer. https://tinyurl.com/ep4sllist