1 Playing with Puppers

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

```java
public class Dog {
    public Dog(){ /* D1 */ }
    public void bark(Dog d) { /* Method A */ }
}

public class Corgi extends Dog {
    public Corgi(){ /* C1 */ }
    public void bark(Corgi c) { /* Method B */ }
    @Override
    public void bark(Dog d) { /* Method C */ }
    public void play(Dog d) { /* Method D */ }
    public void play(Corgi c) { /* Method E */ }
}
```

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.

```java
public static void main(String[] args) {
    Corgi c = new Corgi(); Compile-Error Runtime-Error C1 D1
    Dog d = new Dog(); Compile-Error Runtime-Error C1 D1
    //There is always an implicit call to the superclass's constructor.
    Dog d2 = new Dog(); Compile-Error Runtime-Error C1 D1
    Corgi c2 = new Dog(); Compile-Error Runtime-Error C1 D1
    Corgi c3 = (Corgi) new Dog(); Compile-Error Runtime-Error C1 D1
    //During compile time, we can cast an object along a class's hierarchy with no
    //problem. At runtime, java is upset that the Dog instance "is not" a Corgi. That
    //is, a Dog does not extend from Corgi. However, the dog is instantiated before
    //java attempts to assign it.
    d.play(d); Compile-Error Runtime-Error A B C D E
    d.play(c); Compile-Error Runtime-Error A B C D E
    //d's static type Dog does not have a play method.
    c.play(d); Compile-Error Runtime-Error A B C D E
```
Inheritance and Testing

At compile time, we check c's static type, Corgi, does have a play method that takes in a Dog. At runtime, we look at c's dynamic type, Corgi, for a play method. Here we see play is overloaded, so we pick the method with the "more specific" parameters relative to our arguments, which is method D.

c.play(c); Compile-Error Runtime-Error A B C D E
//Same as previous.

c.bark(d); Compile-Error Runtime-Error A B C D E
c.bark(c); Compile-Error Runtime-Error A B C D E
d.bark(d); Compile-Error Runtime-Error A B C D E
//We notice that bark is overloaded and overridden. As a reminder, dynamic method //selection applies to overridden methods. Method C overrides Method A, and method B //overloads C. For c.bark(c), the compiler had bound caller c's static type's bark to //argument c's most specific static type, Corgi, thus binding method B.

d.bark((int) c); Compile-Error Runtime-Error A B C D E
//During compile time, the compiler will complain that a Corgi "is not" an int. //You can only cast up or down the hierarchy.

c.bark((Corgi) d2); Compile-Error Runtime-Error A B C D E
//During compile time, we check c's static type, Corgi, //for a bark method that takes in a Corgi, which exists, so there is //no compile time error. At runtime, java is upset that d2 //"is not" a Corgi. Note that the cast only temporarily //changes the static type for this SPECIFIC line.

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 chosen at runtime will try to exactly match with our snapshot. The signature consists of the method name, and the number and type of its parameters.
3. During runtime, if call is an overridden 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, IndexOutOfBoundsException, etc).

Notes:
- If a method is overloaded and overridden, 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)

```java
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)

```java
public class SList {
    public class IntNode {
        public int item;
        public IntNode next;
        public IntNode(int i, IntNode n) {
            item = i;
            next = n;
        }
    }
    private static IntNode sentinel; // static sentinel is the flaw
    public SList() {
        sentinel = new IntNode(982734, null);
    }
    public void insertFront(int x) {
        sentinel.next = new IntNode(x, sentinel.next);
    }
    public int getFront() {
        if (sentinel.next == null) {
            return -1;
        }
        return sentinel.next.item;
    }
}
```

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.

```java
@Test
public void test() {
    SList s1 = new SList();
    SList s2 = new SList();
    s1.insertFront(1);
    s2.insertFront(2);
    assertNotEquals(s1.getFront(), s2.getFront());
    assertEquals(1, s1.getFront()); /* also fails */
}
```

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