CS 61B Discussion 5: Exam Prep Spring 2020

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)

```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, circle the options corresponding to the methods that will be executed at **runtime**. If there will be a compiler error or runtime error, circle that instead.

```java
public static void main(String[] args) {
    Dog d = new Corgi();          Compiler-Error   Runtime-Error  C1  D1
    Corgi c = new Corgi();        Compiler-Error   Runtime-Error  C1  D1
    Dog d2 = new Dog();           Compiler-Error   Runtime-Error  C1  D1
    Corgi c2 = new Dog();         Compiler-Error   Runtime-Error  C1  D1
    Corgi c3 = (Corgi) new Dog(); Compiler-Error   Runtime-Error  C1  D1

    d.play(c);           Compiler-Error   Runtime-Error   A   B   C   D   E
    d.play(d);           Compiler-Error   Runtime-Error   A   B   C   D   E
    c.play(d);           Compiler-Error   Runtime-Error   A   B   C   D   E
    c.play(c);           Compiler-Error   Runtime-Error   A   B   C   D   E
    c.bark(d);           Compiler-Error   Runtime-Error   A   B   C   D   E
    c.bark(c);           Compiler-Error   Runtime-Error   A   B   C   D   E
    d.bark(d);           Compiler-Error   Runtime-Error   A   B   C   D   E
    d.bark(c);           Compiler-Error   Runtime-Error   A   B   C   D   E
    d.bark((int)c);      Compiler-Error   Runtime-Error   A   B   C   D   E
    c.bark((Corgi)d2);   Compiler-Error   Runtime-Error   A   B   C   D   E
    ((Corgi)d).bark(c);  Compiler-Error   Runtime-Error   A   B   C   D   E
    ((Dog) c).bark(c);   Compiler-Error   Runtime-Error   A   B   C   D   E
    c.bark((Dog) c);     Compiler-Error   Runtime-Error   A   B   C   D   E
}
```

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.tail`. You may not change anything in the given code. You may only fill in blanks. You may not need all blanks. (Adapted from Spring '17, MT1)

```
public class DMSList {
    private IntList sentinel;
    public DMSList() {
        sentinel = new IntList(-1000, ________________________);
    }
    public class IntList {
        public int head;
        public IntList tail;
        public IntList(int h, IntList t) {
            head = h;
            tail = t;
        }
        public int max() {
            return Math.max(item, tail.max());
        }
    }
    public ______________________________________ {

    ______________________________________________________________________

    ______________________________________________________________________

    ______________________________________________________________________

    ______________________________________________________________________

    ______________________________________________________________________

    ______________________________________________________________________

    ______________________________________________________________________

    ______________________________________________________________________
    }
    /* Returns 0 if list is empty. Otherwise, returns the max element. */
    public int max() {
        return sentinel.tail.max();
    }
}
```

3 Flirbocon

Consider the declarations below. Assume that `Falcon` extends `Bird`. (Spring '17, MT1)

```
Bird bird = new Falcon();
Falcon falcon = (Falcon) bird;
```

Consider the following possible features for the `Bird` and `Falcon` classes. Assume that all methods are **instance methods** (not static!). The notation `Bird::gulgate(Bird)` specifies a method called `gulgate` with parameter of type `Bird` from the `Bird` class.

F1. The `Bird::gulgate(Bird)` method exists.
F2. The `Bird::gulgate(Falcon)` method exists.
F3. The `Falcon::gulgate(Bird)` method exists.
F4. The `Falcon::gulgate(Falcon)` method exists.

(a) Suppose we make a call to `bird.gulgate(bird);`

Which features are sufficient **ALONE** for this call to compile? For example if feature F3 or feature F4 alone will allow this call to compile, select F3 and F4.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Bird::gulgate(Bird)` method. For example, if having features F2 and F4 only (and not F1 and F3) would result in `Bird::gulgate(Bird)` being executed, only select F2 and F4.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Bird)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

(b) Suppose we make a call to `falcon.gulgate(falcon);`

Which features are sufficient **ALONE** for this call to compile?

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Bird::gulgate(Bird)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Bird::gulgate(Falcon)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Bird)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Falcon)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible