CS 61B
Spring 2019

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)

```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.

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

    d.play(d);        Compile-Error   Runtime-Error   A   B   C   D   E
    d.play(c);        Compile-Error   Runtime-Error   A   B   C   D   E
    c.play(d);        Compile-Error   Runtime-Error   A   B   C   D   E
    c.play(c);        Compile-Error   Runtime-Error   A   B   C   D   E
    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
    d.bark((int)c);   Compile-Error   Runtime-Error   A   B   C   D   E
    c.bark((Corgi)d2);Compile-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. 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, _____________________);
    }
    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 _________________________________ {

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________
    }
    /* 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)

```
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;
  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.

```
 @Test
 public void test() {
 ________________________________________________________;
 ________________________________________________________;
 ________________________________________________________;
 ________________________________________________________;
 ________________________________________________________;
 ________________________________________________________;
 ________________________________________________________;
}
```