CS 61B Spring 2021

Topical Review 3

February 17, 2021

1 Doggo

The Dog class is defined for you below. We want to create a Husky class that is a subclass of the Dog class. Huskies are dogs, but they also have a color attribute (default is white). Additionally, they bark in capital letters. They don't bark once; they bark the number of times as the last digit in their weight. For example, if they weigh 47 pounds, they will bark 7 times. We also want to be able to update a Husky's age by passing in age as a variable. Complete the Husky clss below.

```
public class Dog {
    String name;
    int age;
    int weight;

    public Dog() {
        this.name = "Doggo";
        this.age = 0;
        this.weight = 5;
    }

    public void bark() {
        System.out.println("bark");
    }
}
```

```
public class Husky extends Dog {
    String color;

    public Husky() {
        super();
        this.color = "White";
    }

    public void bark() {
        for(int i = 0; i < age % 10; i++) {
            System.out.println("BARK");
        }
    }

    public void updateAge(int age) {
        this.age = age;
    }
}
```

2 ADT Selection

Implement the `SortedList` interface. The interface should support getting an element at a given index, performing an in-place merge with another `SortedList`, and constructing a `SortedList` with one element. You can assume `SortedList`s always contain `int`s.

```
public interface SortedList {
    /* Initialize a SortedList with one element. */
    public SortedList(int elem);

    /* Get the element at index i. */
        public int get(int i);

        /* Merge this list with other. Postcondition: this SortedList must remain in sorted order */
        public void merge(SortedList other);
}
```

(a) Suppose we'd like to perform merge operations between lists using only a constant amount of additional memory. Should `SortedList` be implemented using an internal linked list or an internal array?

Linked list. This would allow us to easily insert elements to a list during the merging process, by setting pointers between the nodes.

(b) Now suppose we'd like to optimize the speed of our `SortedList` data stucture's get operations. Again, select an internal data structure (array or linked list) for `SortedList`.

Array. With an array as the internal data structure, the get method can be implemented in constant time.

3 Interfacitance

Consider this school class:

```
public class School {
    String name;
    int numStudents;

    public void cheer() {
        System.out.println("I have no idea what to say.");
    }

    public void enrollStudent() {
        numStudents += 1;
        if (numStudents % 1000 == 0) {
            System.out.println("We have " + numStudents + " students!");
        }
    }

    public void expelStudent() {
        numStudents -= 1;
    }
}
```

(a) Enrolling and expelling students makes sense but we don't know what a School should do for its cheer. We want subclasses of School to have their own special way to cheer. Suppose we changed School to an interface. Which methods should we make default? Why is a School interface a bad idea?

We would make `enrollStudent` and `expelStudent` but leave `cheer` unimplemented. An interface's fields must be labelled `public final`, so we would never be able to change `numStudents`. We will see later in the course that abstract classes are a better choice here.

(b) We want to create a University class so we can create school instances of different education levels. Oski tried his best, but he didn't take CS61B. University cheers should output the name followed by a space and the motto. Also, Oski forgot that Universities congratulate students upon enrolling them. In addition to doing what enroll currently does, the method should also print "Congratulations!". Fix Oski's University class so it compiles and follows University behaviors.

```
public class University extends School {
String motto; // they should add this. but not add name.

public University(String name, String motto) {
    this.name = name;
    this.motto = motto;
}
```

```
    public void String cheer() {
        // change this to void and removed the return statement.
        String chant = name + ' ' + motto;
        System.out.println(chant);

    }

    public void enrollStudent() {
        // Should ask students to go in this direction instead of copypasting the original method
        super.enrollStudent();
        System.out.println("Congratulations!")
    }
}
```

(c) Stanford thinks they are too cool for school. They wrote their own class following University guidelines. But it's quite unnecessary.

```
public class Stanfurd {
    public void cheer() {
        System.out.println("Stanfurd is 2cool4skool");
    }

    public void enrollStudent() {
        numStudents += 1;
        if (numStudents % 1000 = 0) {
          System.out.println("We have " + numStudents + " students!");
        }
        System.out.println("Congratulations!")
    }

    public void expelStudent() {
        students -= 1;
    }
}
```

Show how simple it is to create a School instance with the same functionality as the Stanfurd class

```
School stanfurd = new University ("Stanfurd", "is 2cool4skool");
```

4 Static Vs. Dynamic Practice

```
public class Fingerprint {...}
public class Key { ... }
public class SkeletonKey extends Key { ... }s

public class StandardBox { public void unlock(Key k) { ... } } // UK

public class BioBox extends StandardBox {
    public void unlock(SkeletonKey sk) { ... } // USK
    public void unlock(Fingerprint f) { ... } // UF
}
```

For each of the lines below, indicate what the output would be (UK, USK, or UF). If there will be a compile-timer error, write CE and if there will be a run-time error, write RE.

```
1  public static void doStuff(Key k, SkeletonKey sk, Fingerprint f) {
2      StandardBox sb = new StandardBox();
3      StandardBox sbbb = new BioBox();
4      BioBox bb = new BioBox();
5
6      sb.unlock(k); UK
7      sbbb.unlock(k); UK
8      bb.unlock(k); UK
9
10     sb.unlock(sk); UK
11     sbbb.unlock(sk); UK
12     bb.unlock(sk); USK
13
14     sb.unlock(f); CE
15     sbbb.unlock(f); CE
16     bb.unlock(f); UF
17
18     bb = (BioBox) sbbb; No Error
19
20     ((StandardBox) bb).unlock(sk); UK
21     ((StandardBox) sbbb).unlock(sk); UK
22     ((BioBox) sb).unlock(sk); RE
23 }
```

5 Dynamic Method Selection with Casting

Suppose we have the following Dog, Corgi, and Retriever classes:

```
public class Dog {
    public void bark() {}
}

public class Corgi extends Dog {
    public void herd() {}
}

public class Retriever extends Dog {
    public void swim {}
}
```

For each line below, write CE if there is a compiler error, RE if there is a runtime error, or nothing if there are no errors.

```
1  public static void main(String[] args) {
2      Dog dog = new Dog();
3      Corgi corgi = new Corgi();
4      Dog bob = new Corgi();
5
6      ((Dog) corgi).bark(); nothing
7      ((Dog) corgi).herd(); CE
8      ((Corgi) corgi).herd(); nothing
9      ((Corgi) dog).bark(); RE
10     ((Corgi) dog).herd(); RE
11     ((Retriever) corgi).swim() CE
12 }
```