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 ______________________________ {

    public Husky() {



    }

    public void bark() {




    }

    public void updateAge( ) {

    }
}
```

2 ADT Selection

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

```
public interface SortedList {









}
```

(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?

(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`.

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() {
        students -= 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?

(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 _______________ {


    public University(String name, String motto) {


    }

    public String cheer() {
        String chant = name + ' ' + motto;
        System.out.println(chant);
        return chant;
    }









}
```

(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

4 Static Vs. Dynamic Practice

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

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.

```
public static void doStuff(Key k, SkeletonKey sk, Fingerprint f) {
    StandardBox sb = new StandardBox();
    StandardBox sbbb = new BioBox();
    BioBox bb = new BioBox();

    sb.unlock(k);   ________
    sbbb.unlock(k); ________
    bb.unlock(k);   ________

    sb.unlock(sk);  ________
    sbbb.unlock(sk);  ________
    bb.unlock(sk);  ________

    sb.unlock(f);   ________
    sbbb.unlock(f);  ________
    bb.unlock(f);   ________

    bb = (BioBox) sbbb; ________

    ((StandardBox) bb).unlock(sk);      ________
    ((StandardBox) sbbb).unlock(sk);    ________
    ((BioBox) sb).unlock(sk);           ________
}
```

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.

```
public static void main(String[] args) {
    Dog dog = new Dog();
    Corgi corgi = new Corgi();
    Dog bob = new Corgi();

    ((Dog) corgi).bark();   ________
    ((Dog) corgi).herd();   ________
    ((Corgi) corgi).herd();   ________
    ((Corgi) dog).bark();     ________
    ((Corgi) dog).herd();     ________
    ((Retriever) corgi).swim()    ________
}
```