Test 1 Review

Discussion 05

Announcements

- Test 1 on Wednesday, 02/16
- Enigma released!

Review

Fun with Methods

Method Overloading is done when there are multiple methods with the same name and return type, but different parameters.

```
public void barkAt(Dog d) { System.out.print(“Woof, it’s another dog!”); }
public void barkAt(Animal a) { System.out.print(“Woof, what is this?”); }
```

Method Overriding is done when a subclass has a method with the exact same function signature as a method in its superclass.

In Dog class:

```
public void speak() { System.out.print(“Woof, I’m a dog!”); }
```

In Corgi Class:

```
public void speak() { System.out.print(“Woof, I’m a corgi!”); }
```

Casting

Casting allows our compiler to overlook cases where we are calling a method that belongs to a subclass on a variable that is statically typed to be the superclass.

```
Animal a = new Dog();
Dog d = a;
Dog d = (Dog) a;
```

Dynamic Method Selection

Your computer…

@ Compile Time:

1. Check for valid variable assignments
2. Check for valid method calls (only considering static type)

@ Run Time:

1. Check for overridden methods
2. Ensure casted objects can be assigned to their variables (only considering dynamic type)

Fields are **always** chosen based on static type!

CS 61B
Spring 2022

Inheritance and Test Review

Exam Prep Discussion 5: February 14, 2022

Note this worksheet is very long and is not expected to be finished in an hour.

1 Athletes

Suppose we have the Person, Athlete, and SoccerPlayer classes defined below.

```java
class Person {
    void speakTo(Person other) { System.out.println("kudos"); }
    void watch(SoccerPlayer other) { System.out.println("wow"); }
}

class Athlete extends Person {
    void speakTo(Athlete other) { System.out.println("take notes"); }
    void watch(Athlete other) { System.out.println("game on"); }
}

class SoccerPlayer extends Athlete {
    void speakTo(Athlete other) { System.out.println("respect"); }
    void speakTo(Person other) { System.out.println("hmph"); }
}
```

$$\begin{tikzcd}
\text{Person} \arrow[d] \\
\text{Athlete} \arrow[d] \\
\text{Soccer Player}
\end{tikzcd}$$

(a) For each line below, write what, if anything, is printed after its execution. Write CE if there is a compiler error and RE if there is a runtime error. If a line errors, continue executing the rest of the lines.

	Static	Dynamic
itai	Person	Person
~~shivani~~	~~Soccer Player~~	~~Person~~
sohum	Athlete	Soccer Player
jack	Person	Athlete
anjali	Athlete	Athlete
chirasree	Soccer Player	Soccer Player

```java
Person itai = new Person();

SoccerPlayer shivani = new Person();CE

Athlete sohum = new SoccerPlayer();

Person jack = new Athlete();

Athlete anjali = new Athlete();

SoccerPlayer chirasree = new SoccerPlayer();

itai.watch(chirasree);void watch (SoccerPlayer other)   "wow"

jack.watch(sohum);CE

itai.speakTo(sohum);void speakTo (Person other)   "kudos"

```

```
19  jack.speakTo(anjali); void speakTo(Person other)   "Kudos"
20
21  anjali.speakTo(chirasree); Void speakTo(Athlete other) "take notes"
22
23  sohum.speakTo(itai); void speakTo(Person other)   "hmph"
                                   ↑ overridden
24
25  chirasree.speakTo((SoccerPlayer) sohum); void speakTo(Athlete other) "respect"
26
27  sohum.watch(itai); CE
28
29  sohum.watch((Athlete) itai); RE
30
31  ((Athlete) jack).speakTo(anjali); void speakTo(Athlete other) "take notes"
32
33  ((SoccerPlayer) jack).speakTo(chirasree); CE  Incorrect cast
34
35  ((Person) chirasree).speakTo(itai); void speakTo(Person other) "hmph"
```

(b) You may have noticed that `jack.watch(sohum)` produces a compile error. Interestingly, we can resolve this error by **adding casting**! List two fixes that would resolve this error. The first fix should print `wow`. The second fix should print `game on`. Each fix may cast either `jack` or `sohum`.

1. jack. watch((Soccer Player) sohum)
2. ((Athlete) jack). watch(sohum)

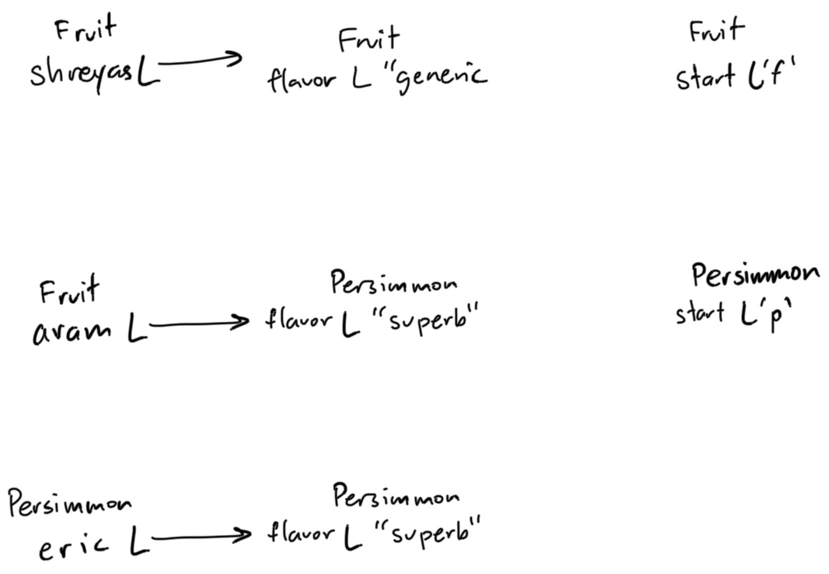
(c) Now let's try resolving as many of the remaining errors from above by **adding or removing casting**! For each error that can be resolved with casting, write the modified function call below. Note that you cannot resolve a compile error by creating a runtime error! Also note that not all, or any, of the errors may be resolved.

Remove casting from line 33: jack. speakTo(chirasree)

2 Hidden Fruits

Suppose we have the `Fruit` and `Persimmon` and classes defined below.

```java
class Fruit {
    String flavor = "generic";
    static char start = 'f';

    static int eat(Fruit fruit) {
        return 1;
    }

    char hats() {
        return this.start;
    }
}

class Persimmon extends Fruit {
    String flavor = "superb";
    static char start = 'p';

    static int eat(Fruit fruit) {
        return 2;
    }

    int eat(Persimmon persimmon) {
        return 3;
    }
}
```



For each line below, write what, if anything, is printed after its execution. Write CE if there is a compiler error and RE if there is a runtime error. If a line errors, continue executing the rest of the lines.

```java
Fruit shreyas = new Fruit();
Fruit aram = new Persimmon();
Persimmon eric = new Persimmon();
                                          No dynamic / static checking
System.out.println(eric.flavor); superb
System.out.println(aram.flavor); generic ← DMS doesn't apply to variables

System.out.println(eric.eat(shreyas)); static int eat (Fruit fruit)        2
System.out.println(eric.eat(eric)); int eat (Persimmon persimmon)          3
System.out.println(aram.eat(eric)); static int eat (Fruit fruit)           1

System.out.println(aram.hats()); char hats()     'f'
System.out.println(eric.hats()); char hats()     'f'
```

3 Containers - I made the walkthrough video for this question.

a) (1 Points). Suppose that we have the `Container` abstract class below, with the abstract method `pour` and the method `drain`. Implement the method `drain` so that all the liquid is drained from the container, i.e. `amountFilled` is set to 0. Return `true` if any liquid was drained, and `false` otherwise. In other words, return true if and only if there is liquid in the container prior to the function being called. You may add a maximum of **5 lines of code**. Note that the staff solution uses 3. You may *only* add code to the `drain` method. (Summer 2021 MT1)

```java
public abstract class Container {
    /* Keeps track of the total amount of liquid in the container */
    public int amountFilled;

    public boolean drain() {
        boolean liquidPrior = amountFilled > 0;
        amountFilled = 0;
        return liquidPrior;


    } // You may use at most 5 lines of code, i.e. this bracket should be on line 11 or earlier.

    abstract int pour(int amount);
}
```

b) (1.5 Points). Finish implementing the `WaterBottle` class so that it **is a** `Container`. You should *only* add code to the blanks, i.e. **fill in the pour method and the class signature**.

As stated in the `Container` class, the `pour` method should pour `amount` into the container and return the amount of the excess liquid, or 0 if there is no excess. For instance, suppose we have a `WaterBottle w` with `capacity` **`10`** and `amountFilled` **`5`**. Then, if we execute `w.pour(7)`, `amountFilled` should be set to **`10`** and **`2`** should be returned. Your solution *must* fit within the blanks provided. You may not need all the lines.

```java
class WaterBottle extends________ Container {
    private static final int DEFAULT_CAPACITY = 16;

    /* The capacity of the container, i.e. the maximum amount of liquid the water bottle can hold */
    private int capacity;

    WaterBottle() {
        this(DEFAULT_CAPACITY);
    }
    WaterBottle(int capacity) {
        this.capacity = capacity;
        this.amountFilled = 0;
    }

    @Override
    public int pour(int amount) {
        amountFilled += amount;  ← can't declare an excess variable here with the lines given and be able to add to amountFilled in time for the if statement
        if (amountFilled > capacity) {
            int excess = amountFilled - capacity;
            amountFilled = capacity;
            return excess;
        }
        return 0;
    }
}
```

c) (4 Points). Finally, suppose we have the `ContainerList` class, with the `drainFirst` method as implemented below. Unfortunately, the `drainFirst` method *sometimes* errors!

In order to fix it, you may add code to the **`ContainerList` constructor and the `UnknownContainer`** class! You may only **use** 5 lines of code in the `ContainerList` constructor and **add** 4 lines of code to the `UnknownContainer` class! If you decide to keep or modify the given line in the `ContainerList` constructor, it counts as one of the 5 lines.

Note that, after making your changes, the `drainFirst` should **never error and retain the functionality in the docstring**. You **may not modify the `drainFirst` method!** You may use classes from the previous part assuming they are implemented correctly.

Hint: Make sure that, with your fix, the `drainFirst` method won't error, even if the `drainFirst` method is called many times.

```
1   class UnknownContainer extends WaterBottle {
2       // TODO                  ↖ Can't extend Container; need WaterBottle to override pour
3       public boolean drain() {
4           return false;
5       }
6
7
8   } // You may add at most 4 lines of code to the class above
9   // i.e. the closing bracket should be on line 6 or earlier
10
11  class ContainerList {
12      private Container[] containers;
13
14      ContainerList(Container[] conts) {
15          this.containers = ~~conts;~~ // you may delete, modify, or keep this line
16          // YOUR CODE HERE new Container[conts.length + 1];
17          for (int index = 0; index < conts.length; index++) {
18              containers[index] = conts[index];
19          }
20          containers[conts.length] = new UnknownContainer();
21
22      } // You may use at most 5 lines of code in the Constructor
23      // i.e. the closing bracket should be on line 18 or earlier
24
25      /* Drains the water from the first nonempty container */
26      void drainFirst() {
27          int index = 0;
28          while (!containers[index].drain()) {
29              index += 1;
30          }
31      }
32  }
```

↑ what if all containers are empty?

The following two problems are very challenging, and we only recommend attempting after finishing the rest of the worksheet.

4 Challenge: Frauds List

(6 Points). Suppose we have the `IntList` and `FraudsList` classes below (Summer 2021, Final)

```java
public class IntList {
    public int first;
    public IntList rest;

    public IntList(int f, IntList r) {
        first = f;
        rest = r;
    }

    public int size() {
        IntList p = this;
        int totalSize = 0;
        while (p != null) {
            totalSize += 1;
            p = p.rest;
        }
        return totalSize;
    }
}

class FraudList extends IntList {
    public FraudList(int f, IntList r) {
        super(f, r);
    }
    public int size() {
        return -super.size();
    }
}
```

Implement the method `findFrauds` which accepts an array of `IntLists` in which some of the elements are, or may contain, `FraudLists`! That is, the dynamic type of certain `IntList` instances is `FraudList`. As shown above, a `FraudList` is an `IntList` whose size method returns the negative of the correct size. You must report these `FraudLists` by **non-destructively** returning a **new** `FraudList` of all the `FraudList` instances linked together in the order they appear in `arr`.

You may **not** modify the given array `arr` or the `IntLists` inside of `FraudList`. You may **not** use `instanceOf`, `getClass()`, `isInstance()` or any method not explicitly written in the classes above or imported. An instance of the problem is shown below:

```
IntList first = new IntList(1000, new IntList(1002, new FraudList(1, new FraudList(2, null))));
IntList second = new FraudList(3, null);
IntList third = new IntList(3000, null);
IntList fourth = new FraudList(4, new IntList(231, new FraudList(5, null)));
IntList[] arr = new IntList[]{first, second, third, fourth};
FraudList frauds = findFrauds(arr);
```

After executing the lines above, `frauds` should be equal to the `FraudList` with the elements `1, 2, 3, 4, 5` and **`arr`, as well as the contents within `arr`, should be unchanged**. Fill in the skeleton below. You may not delete, modify, or add to any of the provided skeleton code.

```
import static java.lang.System.arraycopy;

public static FraudList findFrauds(IntList[] arr) {
    IntList[] copy = new IntList[arr.length];
    arraycopy(arr, 0, copy, 0, arr.length);
    return helper(_copy________, __0_________);
}

public static FraudList helper(IntList[] copy, int index) {
    if (_index == copy.length_________________) {
        return null;
    } else if (_copy[index] == null_____________) {   ← List at the index has been fully looped through
        return helper(copy, index+1)___________;
    }
    IntList current = copy[index]____________________;
    copy[index] = current.rest_______________________;
    if (_current.size() <0__________________) {   ← Checks if the current item is a FraudList
        return new FraudList(current.first, helper(copy, index));
    } else {
        return helper(copy, index)______________;
    }
}
```

5 Challenge: A Puzzle

Consider the **partially** filled classes for A and B as defined below:

```
public class A {
    public static void main(String[] args) {
        _A_ y = new _B_(); Note B y ≠ new A(); means possibilities can be ruled out
        _B_ z = new _B_();
    }

    int fish(A other) {
        return 1;
    }

    int fish(B other) {
        return 2;      overrides
    }
}

class B extends A {
    @Override
    int fish(B other) {
        return 3;
    } Applies if
}     parameter is of
      static type A
```

Note that the only missing pieces of the classes above are static/dynamic types! Fill in the **four** blanks with the appropriate static/dynamic type — A or B — such that the following are true:

1. y.fish(z) equals z.fish(z)=3
2. z.fish(y) equals y.fish(y)=1
3. z.fish(z) does not equal y.fish(y)