Big kid coloring time :)

We'll start at Berkeley time!

Till then, feel free to color this wittle garden using Zoom's annotate feature or ask me anything :)

Iterators, Iterables

Discussion 05

Announcements

Week 5

- ❏ Weekly Survey 4 - due **on Monday 2/15**
- ❏ Lab 4 - due **on Tuesday 2/16**
- ❏ Project 1 - due **on Tuesday 2/16** (only 2 slip days are allowed for this project)
- ❏ Lab 5 - due **on Friday 2/19** (attendance is required)

Content Review

Subtype Polymorphism

Polymorphism in programming describes the ability for methods to work on a variety of types. This gives us more general code, or in other words, a single uniform interface that can work with many types.

Subtype polymorphism is a specific type of polymorphism we achieve from the fact that subtypes of a Class or Interface are also instances of that Class or Interface. Any method that takes in the parent type will take in an instance of the subtype, and so we have polymorphic code. An example:

```
public ComparableArray <T implements Comparable> implements Comparable { ... }
```

Our ComparableArray is polymorphic: it works with any type that is comparable (and we bind our generic to be Comparable items only). Now we can do:

```
T item1 = ...;
T item2 = ...;
item1.compareTo(item2);
```

Exceptions

Exceptions are used to stop the code and inform the person running it when something occurs that is not "allowed".

Examples:

NullPointerException - You tried calling something on a variable that contains null.

IndexOutOfBoundsException - You tried accessing an array index larger than the size of the array.

You can throw your own exceptions:

```
throw new RuntimeException("Because I said so.");
```

You can catch exceptions too (so your code keeps running):

```
try {
    /* Something that may throw an exception */
} catch (Exception e) {
    /* Do something about it */
}
```

Iterators & Iterable

Iterators are objects that can be iterated through in Java (in some sort of loop).

```
public interface Iterator<T> {
    boolean hasNext();
    T next();
}
```

Iterables are objects that can produce an iterator.

```
public interface Iterable<T> {
    Iterator<T> iterator();
}
```

Worksheet

1 Iterators Warmup

```
public interface Iterable<T> {



}

public interface Iterator<T> {



}
```

2A OHQueue

```
1   public class OHRequest {
2       public String description;
3       public String name;
4       public OHRequest next;
5
6       public OHRequest(
                String description,
                String name,
                OHRequest next)
7           this.description =
    description;
8           this.name = name;
9           this.next = next;
10      }
11  }
```

```
import java.util.Iterator;
public class OHIterator _________________________________ {
      OHRequest curr;

      public OHIterator(OHRequest queue) {


      }

      public boolean isGood(String description) {
            return description != null && description.length > 5
      }
```

```
}
```

2B OHQueue

```
1   public class OHRequest {
2       public String description;
3       public String name;
4       public OHRequest next;
5
6       public OHRequest(
                String description,
                String name,
                OHRequest next)
7           this.description =
    description;
8           this.name = name;
9           this.next = next;
10      }
11  }
```

```
import java.util.Iterator;
public class OHQueue _____________________________ {


      public OHQueue(OHRequest queue) {


      }
```

```
}
```

2C OHQueue

```
1   public class OHRequest {
2       public String description;
3       public String name;
4       public OHRequest next;
5
6       public OHRequest(
                String description,
                String name,
                OHRequest next)
7           this.description =
    description;
8           this.name = name;
9           this.next = next;
10      }
11  }
```

```
public class OHQueue ... {
      ...

      public static void main(String[] args) {
            OHRequest s5 = newOHRequest(..., “Allyson”, null);
            OHRequest s4 = newOHRequest(..., “Omar”, s5);
            OHRequest s3 = newOHRequest(..., “Connor”, s4);
            OHRequest s2 = newOHRequest(..., “Hug”, s3);
            OHRequest s1 = newOHRequest(..., “Itai”, s2);


            for (__________________:____________________) {



            }
      }
}
```

3 Thank U, Next

```
public class TYIterator extends ________________________________ {

      public TYIterator(OHRequest queue) {


      }









}
```

4 Senior Class (Extra)

```
public class Person {
    public String name;
    public int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void greet(Person other) {
       System.out.println("Hello, " + other.name);
    }


public class Grandma extends Person {

    public Grandma(String name, int age) {
        super(name, age);
    }

    @Override
    public void greet(Person other) {
        System.out.println("Hello, young whippersnapper");
    }

    public void greet(Grandma other) {
        System.out.println("How was bingo, " + other.name +
"?");
    }
}

```

```
public class testPeople {
    public static void main(String[] args) {
        Person n = new Person("Neil");
        Person a = new Grandma("Ada");
        Grandma v = new Grandma("Vidya");
        Grandma al = new Person("Alex");
        n.greet(a);
        n.greet(v);
        v.greet(a);
        v.greet((Grandma) a);
        a.greet(n);
        a.greet(v);
        ((Grandma) a).greet(v);
        ((Grandma) n).greet(v);
    }
}

```