CS 61BL Lab 12

Ryan Purpura

Exceptions

- There are times where something goes very wrong and you want to break the normal flow of control.
- We've seen certain exceptions before, such as `NullPointerException`
- Exceptions are just objects, and Java defines many useful Exception classes that you can throw; you can also subclass existing exceptions to make your own

Exceptions demo

Exceptions

- Throwing exceptions is done with the `throw` keyword.

```
public static int factorial(int n) {
    if (n < 0) {
        throw new IllegalArgumentException(
            "Factorial argument must be nonnegative"
        );
    }
    if (n == 0) {
        return 1;
    } else {
        return factorial(n - 1) * n;
    }
}
```

Catching Exceptions

- Catching exceptions is done with a try-catch block.
- When an exception is thrown that matches the catch clause, execution is immediately redirected to the body of the catch block.

```
try {
    String in = s.next();
    if (in.equals("quit")) {
        break;
    }
    int x = Integer.parseInt(in);
    System.out.println(factorial(x));
} catch (NumberFormatException e) {
    System.out.println("Bad input: " + e.getMessage());
}
```

Iterators

- Recall the enhanced for (a.k.a. for-each) loop.

```
ArrayList<String> lst = new ArrayList<>();
lst.addAll(List.of("a", "b", "c"));

for (String str : lst) {
    System.out.println(str);
}
```

- The way Java does this is by using an **`Iterator`**
- Iterators are objects that step (aka iterate) through elements of another object

Iterators

- Iterators are classes that implement the Iterator interface.

Interface Iterator<E>

`hasNext()`

Returns `true` if the iteration has more elements.

`next()`

Returns the next element in the iteration.

For Each & Iterators

- The for-each loop internally uses an iterator, something like this:

```
ArrayList<String> lst = new ArrayList<>();
lst.addAll(List.of("a", "b", "c"));

Iterator<String> iterator = lst.iterator();
while(iterator.hasNext()) {
    String str = iterator.next();
    System.out.println(str);
}
```

All iterable classes (like **`ArrayList`**) implement the **`Iterable`** interface, which has a method that returns a new iterator.

```
public interface Iterable<T>
```

`iterator`()

Returns an iterator over elements of type T.

Iterators & Iterables Demo

As a recap:

- To make make a class work with a for-each loop:
  - The class must implement **`Iterable`**
    - The **`Iterable`** interface expects a method named **`iterator()`**, which will return a class that implements Iterator
  - The iterator that gets returned must implement the Iterator interface
    - The Iterator interface expects two methods: one named **`hasNext()`**, and one named **`next()`**