Enums, Generics, Nested Types

June 23, 2017

Reading Quiz

What is true of Generics?

A. They allow you to maintain more type information in your code

B. They are required to store different types of objects in a container

C. They restrict you to having a single type of object in a container

D. They only work with primitive data types.

Whats the Result?

```
ArrayList<Object> parent = new ArrayList<Object>();
Integer child = new Integer(0);

parent.add(child);
System.out.println(parent.size());
```

A. "0"

B. "1"

C. "" (ie empty string)

D. Won't compile

E. Runtime error

Whats the Result?

```
ArrayList<Object> parent = new ArrayList<Integer>();
```

A. Parent is an ArrayList that can contain any type of object

B. Parent is an ArrayList that can contain only Integers (and subtypes)

C. Won't compile

D. Runtime error

Which is true of Nested Types

A. Nested types are a way of describing inheritance

B. Nested types are a way of describing classes with a more convenient syntax

C. Nested types are a way of controlling type visibility

D. Nested types are a way of handing concurrency

Which is **false** of Enums?

A. Enums are a programer convenience that allow you to give your values more meaningful names

B. Enums are a reliability mechanism that allow the type system to catch more errors

C. Enums are a reliability mechanism that allows the type system to perform additional checks

D. Enums are a performance mechanism that allow compiler optimizations

Done!

Housekeeping

- Homework 3 is out
- Class pace
- Feedback

Enums

Problem

- There are cases where we'd like to capture a finite set of values / states
- We'd like the compiler to be able to catch errors
  - Invalid range values
  - Typos

State Range Example

- NetworkRequest.state
  - initialized
  - sent
  - acknowledged
  - receiving
  - finished

State Range Example

- NetworkRequest.state
  - initialized = 0
  - sent = 1
  - acknowledged = 2
  - receiving = 3
  - finished = 4

State Range Example

- NetworkRequest.state
  - initialized = 0
  - sent = 1
  - acknowledged = 2
  - receiving = 3
  - finished = 4

```
switch (aRequest.state) {
  case 0:
    // Send the request
    break;

  case 1:
    // Wait for the request
    break;

  // Other states…
}
```

Issues?

State Range Example

- NetworkRequest.state
  - initialized = "init"
  - sent = "sent"
  - acknowledged = "ack"
  - receiving = "recv"
  - finished = "fin"

```
switch (aRequest.state) {
  case "init":
    // Send the request
    break;

  case "sent":
    // Wait for the request
    break;

  // Other states…
}
```

Issues?

Java Solution

- Enums
- Variables can have a finite number of values
- Values can have human readable names
- Compiler can catch errors

Java Solution

```
public enum NetworkStates {
    INITIALIZED,
    SENT,
    ACKNOWLEDGED,
    RECEIVING,
    FINISHED
}
```

Java Solution

```
public enum NetworkStates {
    INITIALIZED,
    SENT,
    ACKNOWLEDGED,
    RECEIVING,
    FINISHED
}
```

```
switch (aRequest.state) {
  case NetworkStates.INITIALIZED:
    // Send the request
    break;

  case NetworkStates.SENT:
    // Wait for the request
    break;

  // Other states…
}
```

Generics

Problem

- Java's type system is double edged sword
- Specificity can require redundancy
  - Code is written to work with one datatype
  - Rewrite code for other datatypes?
- How to have specificity (types) and convenience (concision)?

Randomizer.java –>

Java Solution: Generics

- Type variables for method signatures
- Ensure consistency of types
- Ex: If you give me an X and a Y, I'll give you a X back
    - String and an Integer
    - Integer and Integer
    - whatever

Randomizer.java –>

Generics in Standard Library

- Extremely common
  - Containers
    - ArrayList<E>
    - Set<E>
    - Map<K, V>
- Documented extensively (ex, ArrayList)

HashExample.java –>

It can get Wild…

- **Functions**
  https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

- **Nested**
  HashMap< HashMap<K, V>, HashMap<V, K> >

HashReverser.java –>

Nested Types

Just the Basics

- `class` is overloaded in Java
  - Types
  - Namespace
  - Data
  - Functionality
- Nested types are a bandaid for this

Problem Example

- Java doesn't have `function` (or anything like it)
- How to pass functionality around?
- Data-less classes
- These are one-offs

NestedTypes.java –>

Homework 3