Programming Languages and Techniques (CIS120)

Lecture 29

 Enums & Iterators
Announcements

• Prof. Benjamin Pierce is lecturing today
  – Will be teaching 120 again next Spring
Enumerations
Enumerations (a.k.a. Enum Types)

- Java supports *enumerated* type constructors.
  - These are a bit like OCaml’s datatypes.

- Example (from PennPals HW):

```java
public enum ServerError {
    OKAY(200),
    INVALID_NAME(401),
    NO_SUCH_CHANNEL(402),
    NO_SUCH_USER(403),
    ...
    // The integer associated with this enum value
    private final int value;

    ServerError(int value) {
        this.value = value;
    }

    public int getCode() {
        return value;
    }
}
```
Using Enums: Switch

// Use of 'enum' in CommandParser.java (PennPals HW)
CommandType t = ...  

switch (t) {
  case CREATE : System.out.println("Got CREATE!"); break;
  case MESG : System.out.println("Got MESG!"); break;
  default: System.out.println("default");
}

- Multi-way branch, similar to OCaml’s match
  - Not pattern matching! (Cannot bind subcomponents of an Enum)

- The default keyword specifies the “catch all” case
What will be printed by the following program?

```
Command.Type t = Command.Type.CREATE;

switch (t) {
    case CREATE : System.out.println("Got CREATE!");
    case MESG : System.out.println("Got MESG!");
    case NICK : System.out.println("Got NICK!");
    default : System.out.println("default");
}
```

1. Got CREATE!
2. Got MESG!
3. Got NICK!
4. default
5. something else
break

- **GOTCHA**: By default, each branch will “fall through” into the next!
  
  Got CREATE!
  Got MESG!
  Got NICK!
  default

- Use an explicit `break` to avoid fallthrough:

```java
switch (t) {
    case CREATE : System.out.println("Got CREATE");
                 break;
    case MESG  : System.out.println("Got MESG");
                 break;
    case NICK  : System.out.println("Got NICK");
                 break;
    default:    System.out.println("default");
}
Enumerations

- Enum types are just a convenient way of defining a class along with some standard methods.
  - Enum types (implicitly) extend java.lang.Enum
  - They can contain constant data “properties”
  - As classes, they can have methods -- e.g. to access a field
  - Intended to represent constant data values

- Automatically generated static methods:
  - `valueOf`: converts a `String` to an `Enum`
    
    ```java
    Command.Type c = Command.Type.valueOf("CONNECT");
    ```
  - `values`: returns an `Array` of all the enumerated constants
    
    ```java
    Command.Type[] varr = Command.Type.values();
    ```
Iterating over collections

iterators, while, for, for-each loops
Iterator and Runnable

interface Iterator<E> {
    public boolean hasNext();
    public E next();
    public void delete(); // optional
}

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

Challenge: given a List<Book> how would you add each book’s data to a catalogue using an iterator?
While Loops

syntax:

```
// repeat body until condition becomes false
while (condition) {
  body
}
```

example:

```
List<Book> shelf = ... // create a list of Books

// iterate through the elements on the shelf
Iterator<Book> iter = shelf.iterator();
while (iter.hasNext()) {
  Book book = iter.next();
  catalogue.addInfo(book);
  numBooks = numbooks + 1;
}
```
For Loops

syntax:

```
for (init-stmt; condition; next-stmt) {
  body
}
```

equivalent while loop:

```
init-stmt;
while (condition) {
  body
  next-stmt;
}
```

List<Book> shelf = … // create a list of Books

// iterate through the elements on the shelf
for (Iterator<Book> iter = shelf.iterator();
     iter.hasNext(); ) {
  Book book = iter.next();
  catalogue.addInfo(book);
  numBooks = numbooks+1;
}
For-each Loops

Syntax:

```
// repeat body for each element in collection
for (type var : coll) {
  body
}
```

- **Array of E or instance of Iterable<E>**
- **element type E**

Example:

```
List<Book> shelf = ... // create a list of books

// iterate through the elements on a shelf
for (Book book : shelf) {
  catalogue.addInfo(book);
  numBooks = numbooks + 1;
}
```
For-each Loops (cont’d)

Another example:

```java
int[] arr = … // create an array of ints

// count the non-null elements of an array
for (int elt : arr) {
    if (elt != 0) cnt = cnt+1;
}
```

For-each can be used to iterate over arrays or any class that implements the `Iterable<E>` interface (notably `Collection<E>` and its subinterfaces).
public static void iteratorExample() {
    List<Integer> nums = new LinkedList<Integer>();
    nums.add(1);
    nums.add(2);
    nums.add(7);

    int numElts = 0;
    int sumElts = 0;
    Iterator<Integer> iter = nums.iterator();
    while (iter.hasNext()) {
        Integer v = iter.next();
        sumElts = sumElts + v;
        numElts = numElts + 1;
    }

    System.out.println("sumElts = " + sumElts);
    System.out.println("numElts = " + numElts);
}

What is printed by iteratorExample()?
1. sumElts = 0  numElts = 0
2. sumElts = 3  numElts = 2
3. sumElts = 10  numElts = 3
4. NullPointerException
5. Something else

Answer: 3
public static void forEachExample() {
    List<Integer> nums = new LinkedList<Integer>();
    nums.add(1);
    nums.add(2);
    nums.add(7);

    int numElts = 0;
    int sumElts = 0;
    for (Integer v : nums) {
        sumElts = sumElts + v;
        numElts = numElts + 1;
    }

    System.out.println("sumElts = " + sumElts);
    System.out.println("numElts = " + numElts);
}
Another Iterator example

```java
public static void nextNextExample() {
    List<Integer> nums = new LinkedList<Integer>();
    nums.add(1);
    nums.add(2);
    nums.add(7);

    int sumElts = 0;
    int numElts = 0;
    Iterator<Integer> iter = nums.iterator();
    while (iter.hasNext()) {
        Integer v = iter.next();
        sumElts = sumElts + v;
        numElts = numElts + v;
    }
    System.out.println("sumElts = " + sumElts);
    System.out.println("numElts = " + numElts);
}
```

What is printed by nextNextExample()?

1. sumElts = 0 numElts = 0
2. sumElts = 3 numElts = 2
3. sumElts = 8 numElts = 2
4. NullPointerException
5. Something else

Answer: 5  NoSuchElementException
Exceptions

Dealing with the unexpected
Why do methods “fail”?

• Some methods expect their arguments to satisfy conditions
  – Input to max must be a nonempty list, Item must be non-null, more
    elements must be available when calling next, ...

• Interfaces may be imprecise
  – Some Iterators don't support the "remove" operation

• External components of a system might fail
  – Try to open a file that doesn't exist

• Resources might be exhausted
  – Program uses all of the computer's memory or disk space

• These are all exceptional circumstances...
  – How do we deal with them?
Ways to handle failure

• Return an error value (or default value)
  – e.g. Math.sqrt returns NaN ("not a number") if given input < 0
  – e.g. Many Java libraries return null
  – e.g. file reading method returns -1 if no more input available
  – Caller is supposed to check return value, but it’s easy to forget
  – Use with caution – easy to introduce nasty bugs!

• Use an informative result
  – e.g. in OCaml we used options to signal potential failure
  – e.g. in Java, we can create a special class like option
  – Passes responsibility to caller, who is forced to do the proper check

• Use exceptions
  – Available both in OCaml and Java
  – Any caller (not just the immediate one) can handle the situation
  – If an exception is not caught, the program terminates
Exceptions

• An exception is an object representing an abnormal condition
  – Its internal state describes what went wrong
  – e.g. NullPointerException, IllegalArgumentException, IOException
  – Can define your own exception classes

• Throwing an exception is an emergency exit from the current context
  – The exception propagates up the invocation stack until it either reaches the top of the stack, in which case the program aborts with the error, or the exception is caught

• Catching an exception lets callers take appropriate actions to handle the abnormal circumstances
private void load(String filename) {
    ImageIcon icon;

    try {
        if ((new File(filename)).exists())
            icon = new ImageIcon(filename);
        else {
            java.net.URL u = new java.net.URL(filename);
            icon = new ImageIcon(u);
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
    ...
}
Simplified Example

class C {
    public void foo() {
        this.bar();
        System.out.println("here in foo");
    }
    public void bar() {
        this.baz();
        System.out.println("here in bar");
    }
    public void baz() {
        throw new RuntimeException();
    }
}

What happens if we do (new C()).foo()?
1. Program stops without printing anything
2. Program prints “here in bar”, then stops
3. Program prints “here in bar”, then “here in foo”, then stops
4. Something else