Introduction to data structures, algorithms, and object-oriented programming

March 30, 2016
Java Parameterized (generic) Types

1. Makes more bugs detectable at compile time

2. Types of objects in a container are specified by arguments inside <>s.

3. Eliminates need for some type-casting.
Parameterized type that removes the complexity of dealing with objects in a container.

ArrayList is implemented as a dynamic array because each item must be accessed quickly.

A dynamic array has an initial fixed size that is incremented (usually doubled) when capacity is exceeded.
**ArrayList<E> Interface**

- boolean add(E): appends item to end of list
- void add(i, E): inserts the item E at position i
- boolean contains(o): Returns true if o in list
- E get(i): Returns element at position i in list
- E indexOf(o): Returns index position of o in list
- E size(): Returns number of objects in list
- E remove(i): Removes and returns element at position i
Abstract Data Types

The term abstract data type, or ADT, refers to a set of possible values and a set of operations on those values, without any specification of how the values are to be represented or how the operations are to be implemented.

There are often several different ways to implement the same abstract data type.
Stacks
The Stack ADT

A stack consists of a sequence of items, which should be thought of as piled one on top of the other like a physical stack of boxes or cafeteria trays.

Only the top item on the stack is accessible at any given time. It can be removed from the stack with an operation called **pop**. An item lower down on the stack can only be removed after all the items on top of it have been popped off the stack.

A new item can be added to the top of the stack with an operation called **push**.
The Stack ADT

Suppose you wanted to implement a Stack of ints with an singly-linked list. Then the operations you would write in the code for the stack list would include:

- int pop(): Removes and returns the top element on the stack.
- void push(int i): Adds a new element to the top of the stack.
- boolean isEmpty(): Returns true if the stack is empty.
- int peek(): Returns the value of the item on the top of the stack.
- int size(): Returns the number of items on the stack.
In the linked list implementation of a stack, the top of the stack is usually the node at the head of the list.

So how would you implement the pop(), push(), peek(), size(), and isEmpty() operations on a stack that is implemented with a linked list of INodes?

How would you implement the stack ADT using an ArrayList?
Stack Interface in Java

- Java interface corresponding to a Stack ADT

- This interface requires the definition of class `StackEmptyException`

```java
public interface Stack {
    public int size();
    public boolean isEmpty();
    public Object peek() throws StackEmptyException;
    public void push(Object o);
    public Object pop() throws StackEmptyException;
}
```
Applications of Stacks

• Direct applications
  – Page-visited history in a Web browser
  – Undo sequence in a text editor
  – Memory used by method calls in the Java Virtual Machine
  – Recursive call return order

• Indirect applications
  – Auxiliary data structure for algorithms
  – Component of other data structures
Method Stack in the JVM

- The Java Virtual Machine (JVM) keeps track of the chain of active methods with a stack.

- When a method is called, the JVM pushes on the stack a frame containing:
  - Local variables and return value
  - Program Counter (PC), keeping track of the statement being executed.

- When a method ends, its frame is popped from the stack and control is passed to the method on top of the stack.

```
main() {
  int i = 5;
  hello(i);
}

hello(int j) {
  int k;
  k = j + 1;
  kitty(k);
}
kitty(int m) {
  ...
}
```
Stack Interface in Java

- Java interface corresponding to a Stack ADT

- This interface requires the definition of class StackEmptyException

```java
class StackEmptyException {}

public interface Stack {
    public int size();
    public boolean isEmpty();
    public Object peek() throws StackEmptyException;
    public void push(Object o);
    public Object pop() throws StackEmptyException;
}
```
Throwing Exceptions

• The Stack interface requires the implementing program to use "throw" to produce a new exception when the user tries to pop or peek at the top of an empty stack. Then the method signature includes a "throws" clause.

• Example of using the throw and throws keywords:

```java
public Object pop() throws StackEmptyException {
    if (isEmpty())
        throw new StackEmptyException("Stack Empty");
    ...
}
```
As soon as the exception is thrown, the flow of control exits from the current method.

So when StackEmptyException is thrown, we exit from method pop() and go to the line of code that called this method.
Try and Catch keywords

A call to pop() in a try statement that uses a catch statement to handle the exception.

```java
public void pop()
{
    try
    {
        Object x = arrayStack.pop();
    }
    catch (StackEmptyException e)
    {
        System.out.println("Stack Empty");
    }
    ...
}
```

No throws clause is necessary on the method signature if the exception is caught (i.e. handled) as shown above.
Exception Handling

• The try and catch blocks mean that we are listening for exceptions that are specified in the catch parameter list.

• The flow of control will go to the catch block if an StackEmptyException occurs. System.out.println inside the catch block will get executed. If no exception the catch block is skipped.

• A catch block can contain anything. It does not have to do only System.out.println. There can also be any number of catch blocks for one try statement ending with a finally statement. Each catch block is looking for a particular type of Exception.

• If somewhere in your method, you throw an exception that is not caught, you need to add a throws clause next to your method name after the parenthesized parameter list.
Exception Handling

A program is said to be *robust* if it can exit gracefully or recover from an error without ending program instead of just presenting a stack trace.

Java provides the Exception class to allow us to "catch" and react to problems during code execution. We can use try...catch blocks to either report errors or handle them:

```
try {
    statements-1
}
catch ( exception-class-name variable-name ) {
    statements-2
}
```

General form of try catch
Exception Handling

An Exception is an actual object that is created by the system when running a Java program. For example, when `Integer.parseInt` is used on a String, it can throw a `NumberFormatException`:

```java
try {
    int score = Integer.parseInt(JOptionPane.showInputDialog("Please enter your score as a whole number"));
    JOptionPane.showMessageDialog(null, "Score: " + score);
} catch (NumberFormatException e) {
    System.out.println("Entry was not an Integer.");
    e.printStackTrace();
}
```
Throwing Exceptions

If you try to pop or peek at an empty Stack, your code could throw an exception defined in the Java API:

```java
throw new IllegalStateException("Can't pop from an empty stack.");
```

Look in the Java API to find out if a method throws a particular type of exception.
Writing your own Exceptions

• The StackEmptyException class.

```java
public class StackEmptyException extends RuntimeException {
    public StackEmptyException(String err) {
        super(err);
    }
}
```

• This would be a good candidate for being an inner class since it is associated with a stack.
Exception Hierarchy

The class Throwable and some of its subclasses.
Using the Exception Hierarchy

BufferedReader newin;
try {
    newin = new BufferedReader(
        (new FileReader(fileName)));
}
catch (Exception e) {
    throw new IllegalArgumentException("Can't open file \"" + fileName + \
        "\" for input.\n" + "(Error :" + e + ")");
}

The Exception class is a superclass of all Exceptions. The author of our textbook says this is not good form, but I took this code from the TextIO.java file.
Using the Exception Hierarchy

BufferedReader newin;
try {
    newin = new BufferedReader
            (new FileReader(fileName) );
}
catch (Exception e) {}