CSCI 112
Tuesday, July 20, 2010

Collections and Stacks

Collections

Collection: A data structure used to hold an unspecified number of data elements.
- Unlike an array, the size does not need to be declared at the declaration.

Two Types of Collections:

Linear: Stores data sequentially.
Non-Linear: Stores data as a network, grid, hierarchy, or has no order at all.

Basic Operations of a Collection

- Add elements
- Remove elements

Collections have a starting size of zero.

Goals of a Collection

Abstraction: Details are hidden.
Encapsulation: Data access is restricted.

Good Questions to Ask When Creating a Collection

- How will the collection operate?
- What will the interface look like?
- What problems will the collection solve?
- How will we implement the interface?
- What are the benefits and penalties to each implementation?

Generics

Generic: The ability to create a placeholder type when the type the user will want is not known.

To do this, we use the generic **T** type.

Java's generics require that T be a class.

Code

```
class Myclass <T> {
```

```
        T a;

        void set(T a){
                this.a = a;
        }
        T get(){
                return a;
        }
}
```

Code

```
public void static main{ String[] args ){
        MyClass<String> x;
        x.set( "Hello" );

        MyClass<Integer> y;
        y.set( new Integer(21) );
}
```

Stack

Stack: a collection

Stacks are **LIFO** data structures

- **LIFO:** Last-In-First-Out, the last data value put on the stack is the first data value off the stack

Applications of the Stack

- Operating Systems: concept known as the Stack Pointer
- The undo button on your text editor.
- The back button on your web browser

Six Stack Operations

- **boolean:** *isEmpty()* : returns true if empty, returns false otherwise
- **boolean:** *isFull()* : returns true if full, returns false otherwise
- **int:** *size()* : returns current number of elements on the stack
- **void:** *push(T data)* : adds a value to the top of the stack
- **T:** *pop()* : removes data from the top of the stack
- **T:** *peek()* : returns top element without removing it from the stack

Stack Implementation

Stack.java

```
class Stack<T>{
	final private int MAX_SIZE = 100;

	T[] elements;

	int top;

	public Stack{
		elements = (T[])new Object[MAX_SIZE];
		top = 0;
	}
	public boolean isEmpty(){
		return ( top == 0 );
	}
	public boolean isFull(){
		return ( top == MAX_SIZE )
	}
	public int size(){
		return top;
	}
	public T peek(){
		if( isEmpty() ){
			System.out.println( "Error: stack empty" );
			System.exit(1);
		}
		return elements[top-1];
	}
	public void push( T data ){
		if( isFull() ){
			System.out.println( "Error: stack is Full." );
			System.exit(1);
		}
		element[top] = data;
		top++;
	}
	public T pop(){
		if( isEmpty() ){
			System.out.println( "Error: stack empty" );
			System.exit(1);
		}
		top--;
		return elements[top];
	}
}
```

StackTest.java

```
public class StackTest{
	public static void main( String[] args ){
		Stack<String> s = new Stack<String>();

		s.push( "Hello." );
		s.push( "Hi." );
		s.push( "Howdy." );
		s.pop()
		System.out.println(s.size());
		System.out.println(s.pop());
	}
}
```