Data Structure

Recitation IV
Topic

- Java Generics
- Java error handling
- Stack
- Lab 2
Java Generics

- The following code snippet without generics requires casting:
  ```java
  List list = new ArrayList();
  list.add("hello");
  String s = (String) list.get(0);
  ```

- When re-written to use generics, the code does not require casting:
  ```java
  List<String> list = new ArrayList<String>();
  list.add("hello");
  String s = list.get(0); // no cast
  ```

- Enabling programmers to implement generic algorithms.

- By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.
The type parameter String within the angle brackets declares the ArrayList to be constituted of String (a descendant of the ArrayList's generic Object constituents).

With generics, it is no longer necessary to cast the third line to any particular type, because the result of `v.get(0)` is defined as String by the code generated by the compiler.
Java Generics

- They allow "a type or method to operate on objects of various types while providing compile-time type safety."
- A common use of this feature is when using a Java Collection that can hold objects of any type, to specify the specific type of object stored in it.
Java Generics

• A generic class is defined with the following format:
  
  class name<T1, T2, ..., Tn> { /* ... */ }

• The type parameter section, delimited by angle brackets (<>), follows the class name. It specifies the type parameters (also called type variables) T1, T2, ..., and Tn.

• A type variable can be any non-primitive type you specify: any class type, any interface type, any array type, or even another type variable.
To reference the generic Box class from within your code, you must perform a generic type invocation, which replaces T with some concrete value, such as Integer, to instantiate this class, use the new keyword, as usual, but place <Integer> between the class name and the parenthesis:

```java
Box<Integer> integerBox = new Box<Integer>();
```
The generic OrderedPair class, which implements the generic Pair interface:

```java
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}

public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() { return key; }
    public V getValue() { return value; }
}

Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
Pair<String, String> p2 = new OrderedPair<String, String>("hello", "world");
```
Java Generics

- You can write a single generic method declaration that can be called with arguments of different types. Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately.

- For example, write a method to find the largest value in:
  - 3, 4, 5
  - 6.6, 7.7, 8.8
  - Apple, pear, orange
public class MaximumTest {

    // determines the largest of three Comparable objects
    public static <T extends Comparable<T>> T maximum(T x, T y, T z) {
        T max = x; // assume x is initially the largest
        if ( y.compareTo( max ) > 0 ){
            max = y; // y is the largest so far
        }
        if ( z.compareTo( max ) > 0 ){
            max = z; // z is the largest now
        }
        return max; // returns the largest object
    }

    public static void main( String args[] ) {
        System.out.printf( "Max of %d, %d and %d is %d\n\n", 3, 4, 5, maximum( 3, 4, 5 ) );

        System.out.printf( "Maxm of %.1f,%.1f and %.1f is %.1f\n\n", 6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ) );

        System.out.printf( "Max of %s, %s and %s is %s\n", "pear", "apple", "orange", maximum( "pear", "apple", "orange" ) );
    }
}
Java Generics

- Following are the rules to define Generic Methods:
- All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type.
- Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
- The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
- A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types not primitive types (like int, double and char).
What is an Exception?

- Exceptional event
- Error that occurs during runtime
- Cause normal program flow to be disrupted
  - Examples
  - Divide by zero errors
  - Accessing the elements of an array beyond its range
  - Invalid input
  - Hard disk crash
  - Opening a non-existent file
  - Heap memory exhausted
What Happens When an Exception Occurs?

1. When an exception occurs within a method, the method creates an exception object and hands it off to the runtime system
   - Creating an exception object and handing it to the runtime system is called “throwing an exception”
   - Exception object contains information about the error, including its type and the state of the program when the error occurred
2. The runtime system searches the call stack for a method that contains an exception handler.

3. When an appropriate handler is found, the runtime system passes the exception to the handler. The exception handler chosen is said to catch the exception.

- If the runtime system exhaustively searches all the methods on the call stack without finding an appropriate exception handler, the runtime system (and, consequently, the program) terminates and uses the default exception handler.
Why we need error handling

codeType readFile {
  initialize errorCode = 0;
  open the file;
  if (theFileIsOpen) {
    determine the length of the file;
    if (gotTheFileLength) {
      allocate that much memory;
      if (gotEnoughMemory) {
        read the file into memory;
        if (readFailed) {
          errorCode = -1;
        } else {
          errorCode = -2
        }
      } else {
        errorCode = -3;
      }
    } else {
      close the file;
      if (theFileDidntClose && errorCode == 0) {
        errorCode = -4;
      } else {
        errorCode = errorCode and -4;
      }
    }
  } else {
    errorCode = -5;
  }
  return errorCode;
}
Error along with RuntimeException & their subclasses are unchecked exceptions.

All other Exception classes are checked exceptions.
Checked exceptions

- Checked exceptions: generally those from which a program can recover & it might be a good idea to recover from such exceptions programmatically.
  - FileNotFoundException
  - ParseException.

- Programmer is expected to check for these exceptions by using the try-catch block or throw it back to the caller.
Unchecked exceptions:

- Unchecked exceptions: These are those exceptions that might not happen if everything is in order, but they do occur
  - ArrayIndexOutOfBoundsException
  - ClassCastException
  - When our program is not doing the right thing ...

- Many applications will use try-catch or throws clause for RuntimeExceptions & their subclasses but from the language perspective it is not required to do so.

- Do note that recovery from a RuntimeException is generally possible but unnecessary for the end programmer to check for such exceptions.
Errors are also unchecked exception, the programmer is not required to do anything with these. In fact it is a bad idea to use a try-catch clause for Errors.

No programmer can guess and can handle error. It depends on dynamically based on architecture, OS and server configuration.

Most often, recovery from an Error is not possible and the program should be allowed to terminate.

- OutOfMemoryError
- StackOverflowError
Best practice

- Throw and catch checked exception, carefully avoid runtime exception and pray for not having error!
What is a stack?

- A stack is a data structure from which one can remove or retrieve only the element most recently inserted, known as **top**.
- Last in first out (LIFO) abstract data type.
Stack Operations

- **Push and Pop**

- **Push**: adds to the top of the list, hiding any items already on the stack, or initializing the stack if it is empty.

- **Pop**: removes an item from the top of the list, and returns this value to the caller. A pop either reveals previously concealed items, or results in an empty list.

- A data-element can be pushed (inserted) only at the from front (stack-head) in the series of elements waiting for an operation and popped (deleted) also from front (stack-head).
Push & Pop

- push 8
- push 14
- push 12
- pop 12
- pop 14
- push 6
- pop 6
- pop 8
Stack Example

Towers of Hanoi
Stack Implementation

- To implement a stack data structure we need ...
  - An area of memory to store the data items
  - A Stack Pointer (SP) register to point to the top of the stack
  - A stack growth convention
  - Some well defined operations: initialize, push, pop
Postfix expression

- Also called Reverse Polish notation
- Calculators employing reverse Polish notation use a stack structure to hold values.
- Expressions can be represented in prefix, postfix or infix notations and conversion from one form to another may be accomplished using a stack.
Postfix expression

- Calculate the following postfix expression:
  - Input: 22 3 + 8 * 2 3 4 * - -
  - Output: ?
public int howMany(String element) {
    int times = 0;
    for (int i = 0; i <= lastIndex; i++) {
        loop and count;
    }
    return times;
}
public boolean uniqInsert(String element)
{
    if contain this element
    do insert
    return true;
    else
    return false;
}
Lab2 24

- public boolean delete(String element)
- First deleteAt(int index)
  - Log[i] = log[i+1], lastIndex--
- Second call deleteAt() at the position you find that equal to element
  - For the whole array
    - if log[i] equal to element
      - deleteAt(i)
      - i--
      - break the loop
- Pay attention to “i” in the for loop
Lab2 25

- public int deleteAll(String element)
- Same as delete, instead of break the loop, count the number of deletions
public String smallest()

- Initiate smallest equal to the first element
- Loop the array
- Compare and update the value in smallest
- Finish looping
Lab2 27

- ArrayList
- Resizable-array implementation of the List interface.
- List<String> list = new ArrayList<String>(100);
- list.get(index)
- Bulid in method, add(), contains(), remove(), indestOf() ...
- Doc: http://docs.oracle.com/javase/1.4.2/docs/api/java/util/ArrayList.html
More…

- ArrayList is a sub class of java collection class
- Use method in Collections!
- Example: frequency
- public static int frequency(Collection<?> c, Object o)
- Returns the number of elements in the specified collection equal to the specified object. More formally, returns the number of elements e in the collection such that (o == null ? e == null : o.equals(e)).
More ...

- More build in methods: Binary search, Min, Max ...