ADVANCED TECHNIQUES

- INTEGRATED DEVELOPMENT ENVIRONMENTS
- JAVA GENERICS
- ARRAYLIST, MAP
- LAMBDAS
INTEGRATED DEVELOPMENT ENVIRONMENTS

- IDEs combine the compiler with the text editor
- Many also provide a symbolic debugger to facilitate finding errors
- Let's try HelloWorld together in one, Eclipse
GENERIC PROGRAMMING

• Generic programming – programming in terms of operations of types only. Any type that satisfies the operational constraints may be used.

• In Java – Multiple methods to do this. Polymorphism (at runtime) and Generics (at compile time)
  • A note on Java…no primitive types can be used in generic programming. This is not true of something like C++
PIECE OF CAKE...JUST TREAT EVERYTHING AS AN OBJECT!

1. `public class GenericArray {`
2. `    Object[] objs;`
3. `    …`
4. `    /* Other stuff.`
5. `    But it is limited because Object doesn’t offer much.
6. `    Still...we can store anything!`
7. `    */`
8. `    …`
9. `}
JAVA GENERICS

• Can be better using “Generics”:

1. public class GenericArray<T> {  //T is an non primitive type
2.    T[] objs;
3.   /* Make assumptions on the operations of T, e.g.,
4.    all T have function draw(). Now any type that
5.    satisfies this requirement may be used, regardless of
6.    inheritance tree.
7.   */
8. }

• Use like:

 GenericArray<String> = new GenericArray<String>();

Types are explicitly written by the programmer
JAVA GENERICS

• Can also be used in functions:
  1. `public static <T, S> int compare(T t, S s) {`
  2.   //make assumptions on the types.
  3.   //Any type that satisfies operation constraints may be used!
  4.   return t.compareTo(s);
  5. }

• Used like:
  1. `MyObject1 a;`
  2. `MyObject2 b; //MyObject1 has function “compareTo(MyObject2)”`
  3. `int c = compareTo(a, b);`

Types are implicitly determined by compiler
DATA STRUCTURES

- Data types specifically designed to have “flexible” storage and to do so efficiently
- Here I define some common ones. CMSC 221 delves into how these would be implemented.
ARRAYLIST

• A “growable” array
• Generic class
• Found in java.util.ArrayList (use import)
• Common functions: add, remove, size, contains
• Can also use related classes Vector, LinkedList

1. import java.util.ArrayList;
2. .../*in the code somewhere*/...
3. ArrayList<String> list = new ArrayList<String>();
4. list.add(“Hello”);
5. list.add(“There”);
6. list.remove(“Hello”);
MAPS

• Associative containers relate pairs of data, referred to as key, value pairs
• Example: student id to student record
• Provides very fast lookup!
• Can use HashMap or TreeMap (remember to import)
• Common functions: put, get, remove, size, containsKey, containsValue, etc.

1. import java.util.HashMap;
2. .../*Somewhere in the code*/...
3. HashMap<Integer, String> h =
   new HashMap<Integer, String>();
4. h.put(4, “JLDiablo”);
5. h.put(2, “HelloWorld!”);
6. String x = h.get(2);
JAVA WILDCARDS

• A very related note, wildcards… “?” represents an unknown type. You can put extends or super constraints on ?, “? extends X” or “? super Y”, then:

```java
1. public static void printArrayList(ArrayList<? extends Object> l) {
2.     for (Object e : l)
3.         System.out.print(e + " ");
4. }
```
LAMBDATA FUNCTIONS

• Nameless functions, written directly where they are used

• Example:
  1. `ArrayList<Integer> numbers = new ArrayList<Integer>();`
  2. `for(int i = 0; i < 1000; ++i)`
  3. `numbers.add((int)(Math.random()*10000));`
  4. `Collections.sort(numbers, (Integer i1, Integer i2) -> i1.compareTo(i2));`
MUCH, MUCH, MORE

- Threading/parallel computation
- Networking
- Databases
- Other libraries (e.g., advanced graphics)
- Etc., etc.