Generic Types in Java

(Ch. 21 in Liang)
**What are ‘Generic Types’ or ‘Generics’?**

**Definition**
- A parameters used to modify class, interface and method definitions at compile time
- **Generic types define ‘macros:’** a given class name replaces the type parameter in the source code
- Basic idea: “search and replace” for types in a class definition

**Syntax**

```
<C> for parameter, use as C elsewhere (C must be a class/interface)

public class Widget <C> { .... } // class definition

Widget<String> widget = new Widget<String>(); // use (instantiation)
```

**Purpose: Avoiding ‘Dangerous’ Polymorphism**
Prevent run-time errors (*exceptions*) due to type errors (*casts*)
Example: Comparable Interface

Prior to JDK 1.5 (and Generic Types):
public interface Comparable {
    public int compareTo(Object o)
}

Comparable c = new Date();
System.out.println(c.compareTo("red"));

JDK 1.5 (Generic Types):
public interface Comparable<T> {
    public int compareTo(T o)
}

Comparable<Date> c = new Date();
System.out.println(c.compareTo("red"));

run-time error

compile-time error
“Raw Types” and Associated Compiler Warnings

Raw Types (for backward compatibility)
Classes with generic type parameters used without the type parameters defined
• e.g. Comparator c ≈ Comparator<Object> c

Recommendation: always set generic type parameters for variable types
• e.g. Comparator<Date> c = new Date();

Compiler Warnings
– javac will give a warning about possibly unsafe operations (type errors) at run-time for raw types
  • use -Xlint:unchecked flag (or, -Xlint:all) for detailed messages.
– javac will not compile programs whose generic types cannot be properly defined
  • e.g. Max.java, Max1.java (Liang)
Overview: Data Structures and Abstract Data Types
Storing Data in Java

Variables

Primitive type (int, double, boolean, etc.)
  - Variable name refers to a memory location containing a **primitive value**

Reference type (Object, String, Integer, MyClass, etc.)
  - Variable name refers to a memory location containing a **reference value** for data belonging to an object

Data Structure

A *formal* organization of a set of data (e.g. variables)

- Arrays: variables of a given type in an integer-indexed sequence
  - int intArray[ ] = {1, 2};  int a = intArray[0];  intArray[1] = 5;

- Objects: data member names used to index variables
  - player.name, player.hits, player.team ...  player.hits = 100;
Abstract Data Types (ADTs)

Purpose
Define interfaces for complex data structures
- Hide *(abstract)* implementation details of operations that query and update
- Operations defined independent of the element type (Java: “generic”)

Some Common ADTs

**List:** Sequence of elements. Elements may be inserted or removed from any position in the list

**Stack:** List with last-in, first-out (LIFO) behaviour (“most recent”)
  e.g. call stack

**Queue:** List with first-in, first-out (FIFO) (“in-order”)
  e.g. lining up at a fast-food restaurant or bank
Tree: Graph with directed edges, each node has one parent (except root), no cycles.
   e.g. Decision tree representing possible moves in a game of tic-tac-toe.

Set: Unordered group of unique items
   e.g. Students in a class, the set of words in a text file

Map: Set of entries, each with unique key and (possibly non-unique) value
   e.g. Student grade sheet: (StudentId, Grade)
   e.g. Frequency of words in a text file (Word, Count)