Generics

Lecture 10

A Simple Component

- Client-side view: Pencil
  ```java
  interface Pencil {
      String toString();
      void setColor(Colors newColor);
      void sharpen(int remove);
  }
  ```

- Implementer's view: LeadedPencil
  ```java
  class LeadedPencil implements Pencil {
      private static final int STD_LENGTH = 10;
      private Colors color;
      private int length;
      ...
  }
  ```

See code listings for full documentation

Motivation: Using Components

- Consider a box that holds a pencil
  - See BoxOfPencil.java
  - Box contains at most one pencil
  - Methods: size, contains, insert, removeAny

- Aside: Notice "coding to the interface"
  - Method signatures contain interface types
  - boolean contains(Pencil target)
  - void insert(Pencil item)
  - Pencil removeAny()

- Specifications also contain this type

Recall: Declared vs Dynamic type

- The dynamic type of these arguments and return values will be a reference to an instance of a class that implements Pencil (e.g., LeadedPencil)

Box of Pencils

- Now consider a box that holds a string
  - See BoxOfString.java

- (Aside: Is it coded to the interface?)

- These two class definitions differ only in:
  - The argument type of contains()
  - The argument type of insert()
  - The return type of removeAny()
  - The types mentioned in specifications

- All the rest is identical!

- BoxOfPencil and BoxOfString are like two instantiations of a generic class definition
  - Parameterized by type (not value)
Example: Generic Box Interface

- **Declaration**
  
  ```java
  interface Box<T> { . . . }
  ```

- In body of interface declaration, T can now be used as a type
  
  ```java
  boolean contains(T target)
  void insert(T item)
  T removeAny()
  ```

- See Box.java

- **Vocabulary:**
  
  - T is called a *naked type*
  - Box (i.e. without < >'s) is called a *raw type*

Example: Generic Implementation

- **Declaration**
  
  ```java
  class PlasticBox<T> implements Box<T> { . . .
  PlasticBox() { . . . }
  }
  ```

- In body of class definition, T can now be used as a type
  
  - In fields
    
    ```java
    private T value
    ```
  - In methods
    
    ```java
    public void insert(T item)
    ```

- See PlasticBox.java

  - Note: Name of constructor in class definition is PlasticBox(), not PlasticBox<T>()

Example: Client Use of Generic

- To use generic type: `classname<type>`

  ```java
  Box<Pencil> bp = new PlasticBox<Pencil>();
  bp.insert(new LeadedPencil());
  Pencil p = bp.remove();
  ```

  // the following are all errors...

  ```java
  String s = bp.remove();
  LeadedPencil p2 = bp.remove();
  ```

  ```java
  Box<Pencil> bp2 = new PlasticBox<String>();
  Box<Pencil> bp3 = new Box<Pencil>();
  ```

Example: Comparable Interface

- Some classes have natural orderings
  
  - e.g. `Integer(3) < Integer(14)`

- `java.lang.Comparable`

  ```java
  public interface Comparable<T> { int compareTo(T o) }
  ```

  - Returns negative integer, 0, or positive integer if this object is <, =, or > argument o

- Typical use
  
  ```java
  if (pl.compareTo(p2) < 0) // p1 < p2
  if (pl.compareTo(p2) == 0) // p1 == p2
  if (pl.compareTo(p2) > 0) // p1 > p2
  ```
**Good Practice: Total Ordering**

- compareTo should induce a total ordering on its type parameter
  - Reflexive
    \[ x \text{.compareTo}(x) == 0 \]  
  - Transitive
    \[ x \text{.compareTo}(y) < 0 \&\& y \text{.compareTo}(z) < 0 \Rightarrow x \text{.compareTo}(z) < 0 \]  
  - Antisymmetric
    \[ x \text{.compareTo}(y) \leq 0 \&\& y \text{.compareTo}(x)\leq0 \Rightarrow x \mathsf{equals}(y) \]  
  - Total
    \[ \text{Any two instances of T can be compared} \]

**Implementing Comparable**

- Simple case for typical use
  ```java
  class LeadPencil implements Pencil, Comparable<LeadPencil> {
      int compareTo(LeadPencil o) { . . . }
  }
  ```

- Or even better (coding to the interface!)
  ```java
  class LeadPencil implements Comparable<Pencil> {
      int compareTo(Pencil o) { . . . }
  }
  ```

- Or even better (but we’ll talk about extends later)
  ```java
  interface Pencil extends Comparable<Pencil> { ... }
  class LeadPencil implements Pencil {
      int compareTo(Pencil o) { . . . }
  }
  ```

**Example: Lists**

- Array size fixed by instantiation with new
  ```java
  Integer[] A = new Integer[145];
  ```

- What if you need the array to grow?
  - Allocate new (larger) array
  - Copy old values into new

- Better approach: `java.util.List<T>`
  - Generic interface
  - Holds a (ordered) list of Ts
  - Can be accessed by index like an array
  - But also has a dynamically changeable size

- Implementations: `ArrayList`, `LinkedList`, `Vector`
  - `ArrayList` more efficient, need `Vector` for threads

**Using List (and ArrayList)**

```java
import java.util.List;
import java.util.ArrayList;

List<String> list = new ArrayList<String>();
list.add("Hello");
list.add("there");
list.add(0, "Sam");
System.out.println(list.get(1)); //"Hello"
for (String str : list) {
    System.out.println(str);
} //prints "SamHellothere"
```

**Methods**

- Array-like
  - set / get for index-based access
- Adding items
  - add(T) / add(int,T)
  - Causes the List to grow
- Removing items
  - remove(int) / removeRange(int,int)
- Memory management
  - isEmpty / size

**Type Erasure**

- Note: `PlasticBox<Pencil>` and `PlasticBox<String>` are not two separate classes
  - They are two generic type invocations of one class, `PlasticBox`
  ```java
  PlasticBox b1 = new PlasticBox<Pencil>();
  PlasticBox b2 = new PlasticBox<String>();
  assert b1.getClass() == b2.getClass(); //passes
  ```

- Think of `<Pencil>` as constructor information, so the compiler can do appropriate casting and type checking

- At run-time, no generic type information remains in `PlasticBox` objects
  - The type parameter, T, has been “erased”
  - Left with one class: `PlasticBox<T>`
    (pronounced “plastic box of unknown”)
Box of Pencils

- implements
- uses

**Box<?>**

- PlasticBox<?>
- Pencil
- LeadedPencil

Consequences of Type Erasure

- All type-instances share the same static members
  - static int nextID; //shared by all Box<?>
- Static members can not refer to naked type
  - private static T value; //compile error
- New instances and arrays of naked type can not be created
  - T value = new T(); //compile error
  - T[] myArray = new T[50]; //compile error
- Casts ignore parameter type information
  - Box<String> x = (Box<String>) b; //unchecked
  - Box<?> y = (Box<?>) b; //ok

Summary

- Genericity through type parameters
  - Declaration of generic interfaces/classes
  - Use of generic interfaces/classes
- Comparable interface
  - Total ordering, strongly typed thanks to generics
- List (and ArrayList)
  - Like arrays, but better!