Generics: Java's ArrayList

31 March 2008
CMPT166
Dr. Sean Ho
Trinity Western University

Generics

- We can write classes to define our own ADTs; each instance has attributes of certain type
- But what if we want our ADT to be flexible to handle different types?
- Generics: let type be a parameter
  - When instantiating, specify a type name
- e.g., ArrayList: resizeable array of objects
  - Specify the type of the objects

Example of generics: ArrayList

- Regular Java arrays have a fixed length
  - Can allocate at run-time, but once allocated they stay the same size

```
String[] appleBin = new String[ 10 ];
```

- The class ArrayList expands as necessary

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

  - Instantiate specifying the <element type>:

```
ArrayList<String> appleBin
        = new ArrayList<String>( 10 );
```

  - creates a new empty list of Strings
  - initially with enough space for 10 elements

ArrayList methods

- Add a new element to the array:
    - ◆ appleBins.add(“Fuji”);
  - appends to list, expanding list as needed
    - ◆ appleBins.add(0, “Gala”);
  - inserts at given position, shifting rest of list
- Accessor set/get methods:
    - ◆ appleBins.get(0); // returns “Gala”
    - ◆ appleBins.set(0, “Spartan”);
  - set() can only modify what has previously been add()ed

More ArrayList methods

- Shrink list:
  - appleBin.remove(idx);
  - appleBin.removeRange(start, end); // s ≤ idx < e
  - appleBin.remove(“Fuji”);
    - searches for object and removes first found instance
- Search through list:
  - appleBin.contains(“Fuji”); // boolean
  - appleBin.indexOf(“Fuji”);

for-each works with ArrayList

- ArrayList is a kind of collection
- Iterate over collections with for-each loop:

```
for ( String appleName : appleBin ) {
    System.out.println( "I have a " + appleName + " apple." );
```

ArrayList memory management

- Check current size of list:
    - appleBin.size();
    - appleBin.isEmpty();
- Increase list's capacity:
    - appleBin.ensureCapacity(100);
  - Faster if about to add() lots
- Shrink list to free up unused space:
    - appleBin.trimToSize();
- Shallow copy:
    - appleBin.clone();

Generic/parameterized classes

- You can write your own generic class
  - Specify generic type in angle brackets

    -
      ```
      public class FruitBasket<T> {
          private T item;
          public FruitBasket() { item = null; }
          public FruitBasket( T newitem ) { item = newitem; }
          public T get() { return item; }
          public String toString() { return( "This basket has a "
                  + item.toString() + "." ); }
      ```
  - Instantiate:

    -
      ```
      FruitBasket<Pear> myBasket =
          new FruitBasket<Pear>();
      ```

Tips on generics

- Constructor name doesn't take type parameter:

```
public FruitBasket<T>( T newitem )  // wrong!
public FruitBasket( T newitem )     // right
```

- Type parameters cannot be primitive types:
  - `FruitBasket<int> myBasket        // wrong!`
  - Use the wrapper classes instead
    - `FruitBasket<Integer> myBasket    // right`
- Cannot use constructors of type parameters:
  - `item = new T();      // illegal`
- Generics can have multiple type parameters:
  - `public class FruitBasket<S, T> {`

Type parameter bounds

- Should I allow FruitBasket's item to be any type?
- Specify what superclass or interfaces are needed:

  -
    ```
    public class FruitBasket<
        S extends Fruit,
        T extends Fruit & ActionListener & Runnable>
    ```
  - When instantiating, the chosen types must be subclasses of the given superclasses
    - And must implement the given interfaces
  - This allows our code to use methods from the given classes/interfaces:
    - `T item; item.start();`

Generic subclasses

- A generic class can be a subclass of a generic superclass:
    - `public class FruitBasket<T> extends Basket<T> {`
  - So FruitBasket<Pear> counts as a subclass of Basket<Pear>
    - Type parameters match
  - However, if Pear is a subclass of Fruit, FruitBasket<Pear> is not a subclass of FruitBasket<Fruit>
    - Type parameters must match exactly