

CSCI 136

Data Structures &

Advanced Programming

Generic Types

Using Generic (Parameterized) Types

- What limitations are associated with casting Objects as they are added and removed from Associations?
 - Errors cannot be detected by compiler
 - Must rely on runtime checks
- Instead of casting Objects, Java supports using generic or parameterized data types (Read Ch 4)
- Instead of:

```
Association a = new Association("Bill", 97);
int grade = (Integer) a.getValue(); //Cast to Integer
```

- Use:

```
Association<String, Integer> a =
    new Association<String, Integer>("Bill", (Integer) 97);
Integer grade = a.getValue(); //no cast!
```

Generic Association<K,V> Class

```
class Association<K,V> {
    protected K theKey;
    protected V theValue;

    //pre: key != null
    public Association (K key, V value) {
        Assert.pre (key != null, "Null key");
        theKey = key;
        theValue = value;
    }

    public K getKey() {return theKey;}
    public V getValue() {return theValue;}
    public V setValue(V value) {
        V old = theValue;
        theValue = value;
        return old;
    }
}
```

A First Generic Data Structure

- To illustrate the use of generic types, we'll implement a generic Bag data structure
- A *Bag* is a common structure for holding a collection of objects supporting the operations
 - Add : Add an item to the Bag
 - Remove : Remove some item from the bag
 - The user can't specify which item
 - A Bag can contain duplicate items
 - It's a *multi-set*

Generic Bag Structure

```
public class Bag<E> {  
  
    private Object[] theBag;  
  
    // The bag is filled starting at index 0  
    // nextFreeSlot is location of the first empty slot in bag  
    private int nextFreeSlot;  
  
    // Create a bag of a given capacity  
    public Bag(int capacity) {  
        theBag = new Object[capacity];  
        nextFreeSlot = 0;  
    }  
  
    public boolean isFull() { return nextFreeSlot == theBag.length;}  
  
    public boolean isEmpty() { return nextFreeSlot == 0;}  
  
    // continued...
```

Generic Bag Structure

```
// Add an item in an available location and return true
// Or bag is full so return false
public boolean add(E item) {
    if(isFull()) return false;

    theBag[nextFreeSlot] = item;
    nextFreeSlot++;
    return true;
}

// Remove an item from the bag and return it
public E remove() {
    ifisEmpty()) return null;

    nextFreeSlot--;
    result = (E) theBag[nextFreeSlot];
    theBag[nextFreeSlot] = null;
    return result;
}
```