COSC 2006: Data Structures I

Java 5 Generic Types

For classes in Main Chapter 5

IntPair objects (1)

```
/**
 * A class representing pairs of integers using Object
 */
public class IntPair
{
   private int first;
   private int second;

   public Pair(int first, int second)
   {
      this.first = first;
      this.second = second;
   }
```

IntPair objects (2)

```
    public int getFirst()
    {
        return first;
    }

    public int getSecond()
    {
        return second;
    }

    public String toString()
    {
        return "(" + first + "," + second + ")";
    }

};
```

Object References

- Object references can point to any type of object.
  - String s = “Hello World”;
  - Object o = s;

  - Hero h = new Hero();
  - Object o = h;
- Since Object references are most general, this is a widening conversion.

Narrowing Conversions

- Narrowing conversions require typecasting
  - String s = “Hello World”;
  - Object o = s;
  - String s2 = (String) o; //o is too wide to
                                        //be stored in a String
  - Hero h = new Hero();
  - Object o = h;
  - Hero h2 = (Hero) o; //o is too wide

Non-Generic Pair objects (1)

```java
package nonGeneric;
/**
 * A class representing pairs of objects using Object
 * types.
 */
public class Pair
{
   private Object first;
   private Object second;

   public Pair(Object first, Object second)
   {
      this.first = first;
      this.second = second;
   }
```

Non-Generic Pair objects (2)

```
    public Object getFirst()
    {
       return first;
    }

    public Object getSecond()
    {
       return second;
    }

    public String toString()
    {
       return "(" + first + "," + second + ")";
    }

};
```

Using Pair objects

Define pairs of strings

```
Pair p1 = new Pair("Fred", "Jones");
System.out.println(p1);
```

Use getFirst and getSecond methods (typecasting is necessary)

```
String first = (String) p1.getFirst();
String second = (String) p1.getSecond();
System.out.println(first);
System.out.println(second);
```

Narrowing conversions

Define pairs of names and ages

```
Pair p2 = new Pair("Fred", 18); // autoboxing for 18
String name = (String) p2.getFirst();
int age = (Integer) p2.getSecond()
System.out.println("Name = " + name + ", age = " + age);
```

Generic Pair objects (1)

```
package generic;
/**
 * A class representing pairs of objects of generic types.
 * @param <U> the type of the first object in pair
 * @param <V> the type of the second object in pair
 */
public class Pair<U,V>
{
   private U first;
   private V second;

   public Pair(U first, V second)
   {
      this.first = first;
      this.second = second;
   }
```

Generic Pair objects (2)

```
    public U getFirst()
    {
        return first;
    }

    public V getSecond()
    {
        return second;
    }

    public String toString()
    {
        return "(" + first + "," + second + ")";
    }

};
```

Using Pair objects (1)

Define pairs of strings   NOTE: Must specify the type explicitly

```
Pair<String,String> p1 =
   new Pair<String,String>("Fred", "Jones");
System.out.println(p1);
```

Define pairs of strings & Integers (autoboxing/unboxing is used)

```
Pair<String,Integer> p2 =
   new Pair<String,Integer>("Fred", 18);
int age = p2.getSecond();
System.out.println(p2);
```

Without autoboxing/unboxing we would need to use

```
Pair<String,Integer> p2 =
   new Pair<String,Integer>("Fred", new Integer(18));
int age = p2.getSecond().intValue();
System.out.println(p2);
```

Using Pair objects (2)

Define pairs of doubles

```
Pair<Double,Double> p1 =
   new Pair<Double,Double>(1.23, 3.14);
System.out.println(p1);
```

Note that Pair<double,double> or Pair<int, int> are illegal since only Object types can be used.

COSC 2006: Data Structures I

ArrayBag

using the Java 5 generic type

ArrayBag of strings

ArrayBag is a generic type version of ArrayBag (Object type, 2nd ED). When the bag is constructed we declare using the type identifier that the bag can only hold strings. No typecasting is needed in this example.

```
ArrayBag<String> bag = new ArrayBag<String>();
bag.add("Tom");
bag.add("Dick);
bag.add("Harry");

String s = bag.grab();
System.out.println("Bag is " + bag);
System.out.println("Grabbed " + s);

bag.add(new Point(1,2));
```

Typecast is not necessary here

This is now a compiler error

ArrayBag data fields

We now use an array of generic type E [ ]

```
public class ArrayBag<E> implements Cloneable
{
    private E[] data;
    private int manyItems;
    private static final int
                               INITIAL_CAPACITY = 10;

    ...
```

Array of object type E [ ]

E is a type identifier for the bag elements

Constructing generic arrays (1)

- It is not possible to construct an array of objects of type E directly. There are technical reasons for this (type erasure)
- For example, the following is illegal
  - `data = new E[initialCapacity];`
- It is necessary to use the type cast
  - `data = (E[]) new Object[initialCapacity];`
- This causes a compiler warning which can be safely ignored in this case.

Constructing generic arrays (2)

- This is not a problem since when the expression

```
  new Object[initialCapacity]
```

  is executed there are no objects stored in the array yet so there is no possibility at this stage of having objects of different types in the array. (warning can be turned off)

ArrayBag constructors

Typecast the Object array created to type E [ ]

```
public ArrayBag(int initialCapacity)
{
   if (initialCapacity < 0)
      throw new IllegalArgumentException("...");

   data = (E[]) new Object[intialCapacity];
   manyItems = 0;
}

public ArrayBag()
{ this(INITIAL_CAPACITY);}
```

This will give a compiler warning but ignore it

ArrayBag add method

The element type is now E instead of type Object

```
public void add(E element)
{
    if (manyItems == data.length)
    {   // bag is full so expand it
        ensureCapacity(manyItems*2 + 1);
    }
    data[manyItems] = element;
    manyItems++;
}
```

ArrayBag addMany method

The element type is now E instead of type Object.
Using a variable number of arguments is a new feature of Java 5

```
public void addMany(E... elements)
{
  if (manyItems + elements.length > data.length)
   {
     ensureCapacity(
     (manyItems + elements.length)*2);
   }
   System.arraycopy(
      elements,0, // all of the elements array
      data,manyItems, // start after data end
      elements.length); // copy this many items
   manyItems += elements.length;
}
```

ArrayBag addAll method

Here we just change the parameter type to ArrayBag<E>

```
public void addAll(ArrayBag<E> addend)
{
   if (addend == null)
      throw new IllegalArgumentException("...");

   ensureCapacity(manyItems + addend.manyItems);
   System.arraycopy(
      addend.data, 0, // all of addend array
      data, manyItems,// start after end of data
      addend.manyItems // copy this many items
   )
   manyItems += addend.manyItems; // update
}
```

ArrayBag clone method

Since clone returns Object we type cast it to type ArrayBag<E> instead of Object [ ]. There will be a compiler warning for the typecast but we can ignore it. Can also use the proper return type instead of Object in Java 5 (covariant return type)

compiler warning

```
public ArrayBag<E> clone()
{   ArrayBag<E> answer;
    try
    {   answer = (ArrayBag<E>) super.clone();
    } catch (CloneNotSupportedException e)
    {
        throw new RunTimeException("...");
    }
    answer.data = data.clone();
    return answer;
}
```

countOccurrences method (1)

Change the target type to E

```
public int countOccurrences(E target)
{   int answer = 0;
    if (target == null)
    { // count the number of nulls in bag
       for (int index = 0; index < manyItems;
                                         index++)
         if (data[index] == null)
             answer++
    } else {
       for (int index = 0; index < manyItems;
                                         index++)
         if (target.equals(data[index]))
             answer++;
    }
}
```

ArrayBag ensureCapacity

Change Object type to E and type cast.
For array construction with new the type must be Object

```
public void ensureCapacity(int minimumCapacity)
{
   E[] biggerArray;
   if (data.length < minimumCapacity)
   {
      biggerArray =
         (E[]) new Object[minimumCapacity];
      System.arraycopy(data, 0, biggerArray,
         0, manyItems);
      data = biggerArray;
   }
}
```

compiler warning

ArrayBag grab method

Change the return type to E
array index goes from 0 to manyItems - 1

```
public E grab()
{
   if (manyItems == 0)
      throw new IllegalStateException("...");

   int i = (int)(Math.random() * manyItems);
   return data[i];
}
```

ArrayBag remove method

Change the target type to E

```
public boolean remove(E target)
{   int i = 0;
    if (target == null)   //search for null entries
    {   while ((i < manyItems) && (data[i] != null))
            i++;
    }
    else
    {   while ((i < manyItems) && (!target.equals(
            data[i]))) i++;
    }
    if (i == manyItems)
        return false;
    else
    {   manyItems--; data[i] = data[manyItems];
        data[manyItems] = null; return true;
    }
}
```

ArrayBag trimToSize

Change the array type to E [ ] except with new we must use the Object type

```
public void trimToSize()
{
   E[] trimmedArray;
   if (data.length != manyItems)
   {
      trimmedArray =
         (E[]) new Object[manyItems];
      System.arraycopy(data, 0, trimmedArray,0,
         manyItems);
      data = trimmedArray;
   }
}
```

compiler warning

ArrayBag union method

Since this is a static method it doesn't have access to the class type parameter so it is neccesary to include <E> in prototype

```
public static <E> ArrayBag<E> union(
             ArrayBag<E> b1, ArrayBag<E> b2)
{   if (b1 == null)
       throw new IllegalArgumentException("...");
    if (b2 == null)
       throw new IllegalArgumentException("...");
    ArrayBag<E> answer = new ArrayBag<E>(
       b1.getCapacity() + b2.getCapacity());
    System.arraycopy(b1.data, 0, answer.data, 0,
       b1.manyItems);
    System.arraycopy(b2.data, 0, answer.data, b1.manyItems,
       b2.manyItems);
    answer.manyItems = b1.manyItems + b2.manyItems);
    return answer;
}
```

ArrayBag toString method

Just change the string name

```
public String toString()
{   StringBuilder s = new StringBuilder();
    s.append("ArrayBag[");
    for (int k = 0; k < size(); k++)
    {
        s.append(data[k]);
        // don't put comma after last element
        if (k != size() - 1)
            s.append(",");
    }
    s.append("]");
    return s.toString();
}
```

COSC 2006: Data Structures I

Node, LinkedBag

generic types (Ch5, 3rd ED).

COSC 2006: Data Structures I

GTNode, GTLinkedBag

These are our slightly different versions of the ones in Chapter 5, 3rd ED.

Converting Node to GTNode (1)

- We need a generic node class called GTNode whose data part is an object of type E
- Recall in the Node class (nodes of type Object) that the static methods listCopyWithTail and listPart need to return two nodes. This was done using an array of two nodes
- This is not the OOP way

Converting Node to GTNode (2)

- Instead of an array of two GTNode objects we will use an inner class called Pair to represent two GTNode objects.
- We can then return an object of this class instead of an array of two elements

GTNode data fields

The data field is now of type E
The link field is now of type GTNode<E>

```
public class GTNode<E>
{
   private E data;
   private GTNode<E> link;

   // ...
```

GTNode Constructor

The data parameter is now of type E
The link parameter is now of type GTNode<E>

```
public GTNode(E initialData,
   GTNode<E> initialLink)
{
   data = initialData;
   link = initialLink;
}
```

GTNode NodeAfter methods

The data parameter is now of type E
The new node is of type GTNode<E>

```
public void addNodeAfter(E element)
{
    link = new GTNode<E>(element, link)
}

public void removeNodeAfter()
{
    link = link.link;
}
```

GTNode getData, getLink

The data is now of type E
The link is now of type GTNode<E>

```
public E getData()
{
    return data;
}


public GTNode<E> getLink()
{
    return link;
}
```

GTNode toString

Use type GTNode<E> instead of Node

```
public String toString()
{   StringBuilder s = new StringBuilder();
    s.append("GTNode[");
    GTNode<E> current = this;
    while (current != null)
    {   s.append(current.data);
        if (current.link != null)
        {   s.append(",");
        }
        current = current.link;
    }
    s.append("]");
    return s.toString();
}
```

GTNode listCopy

Use type GTNode<E> instead of Node

```
public static <E> GTNode<E> listCopy(
                              GTNode<E> source)
{   if (source == null) return null;
    GTNode<E> copyHead
       = new GTNode<E>(source.data, null);
    GTNode<E> copyTail = copyHead;

    while (source.link != null)
    {   source = source.link;
        copyTail.addNodeAfter(source.data);
        copyTail = copyTail.link;
    }
    return copyHead;
}
```

GTNode listCopyWithTail

Use type GTNode<E> instead of Node and use Pair object

```
public static <E> Pair<E> listCopyWithTail(
                                GTNode<E> source)
{   if (source == null)
        return new Pair<E>(null, null);
    GTNode<E> copyHead =
        new GTNode<E>(source.data, null);
    GTNode<E> copyTail = copyHead;
    while (source.link != null)
    {   source = source.link;
        copyTail.addNodeAfter(source.data);
        copyTail = copyTail.link;
    }
    return new Pair<E>(copyHead, copyTail);
}
```

GTNode listLength

Use type GTNode<E> instead of Node

```
public static <E> int listLength(GTNode<E> head)
{
   int answer = 0;
   for(GTNode<E> cursor = head; cursor != null;
                                cursor = cursor.link)
   {
      answer++;
   }
   return answer;
}
```

GTNode listPart (1)

Use type GTNode<E> instead of Node and use Pair object

```
public static <E> Pair<E> listPart(
                  GTNode<E> start, GTNode<E> end)
{
   if (start == null)
      throw new IllegalArgumentException("...");
   if (end == null)
      throw new IllegalArgumentException("...");

   // continued
```

GTNode listPart (2)

Use type GTNode<E> instead of Node and use Pair object

```
    GTNode<E> copyHead =
       new GTNode<E>(start.data, null);
    GTNode<E> copyTail = copyHead;
    GTNode<E> cursor = start;
    while (cursor != end)
    {  cursor = cursor.link;
       if (cursor == null)
          throw new
             IllegalArgumentException("...");
       copyTail.addNodeAfter(cursor.data);
       copyTail = copyTail.link;
    }
    return new Pair<E>(copyHead, copyTail);
}
```

GTNode listPosition

Use type GTNode<E> instead of Node

```
public static <E> GTNode<E> listPosition(
   GTNode<E> head, int position)
{
   if (position <= 0)
      throw new IllegalArgumentException("...");
   GTNode<E> cursor = head;
   for (int i = 1; (i < position) &&
                  (cursor != null); i++)
      cursor = cursor.link;
   return cursor;
}
```

NOTE: positions begin at 1 here

GTNode listSearch (1)

Use type GTNode<E> instead of Node

```
public static <E> GTNode<E> listSearch(
   GTNode<E> head, E target)
{
   if (target == null)
   {  for (GTNode<E> cursor = head;
             cursor != null; cursor = cursor.link)
        if (cursor.data == null)
            return cursor;
   }
   else // continued next slide
```

GTNode listSearch (2)

Use type GTNode<E> instead of Node

```
    {
       for (GTNode<E> cursor = head;
          cursor != null; cursor = cursor.link)
       if (target.equals(cursor.data))
          return cursor;
    }

    return null;
}
```

GTNode set methods

Use type GTNode<E> instead of Node and use data type E

```
public void setData(E newData)
{
    data = newData;
}

public void setLink(GTNode<E> newLink)
{
    link = newLink;
}
```

Pair inner class

Use a static class since it doesn't access the outer class.
Outside GTNode<E> class access it as GTNode.Pair<E>

```
public static class Pair<T>
{
   private GTNode<T> head;
   private GTNode<T> tail;

   public Pair(GTNode<T> head, GTNode<T> tail)
   {  this.head = head;
      this.tail = tail;
   }

   public GTNode<T> getHead() { return head; }
   public GTNode<T> getTail() { return tail; }
}
```