Generics

- Make Your Own Dynamic List
- Generics
- Autoboxing

Introduction to Java

See Also

https://docs.oracle.com/javase/tutorial/java/generics/

http://www.tutorialspoint.com/java/java_generics.htm

https://www.youtube.com/watch?v=CEa-srm_l28&list=PL655DCAD6C31F1DDD

Dynamic List of Points

- Create an object that collects Point objects
- Starts out with none
- Methods to add, get, set, insert, remove
- Test-Driven development:
  - Write the testing code first
  - Write the code to pass the tests

```
static void printPoints(DynamicList list) {

    int size = list.getSize();
    for(int x=0;x<size;++x) {
        Point p = list.getElement(x);
        System.out.println(p);
    }

}
```

```
Point a = new Point(1,1);
Point b = new Point(2,2);
Point c = new Point(3,4);

DynamicList list = new DynamicList();
printPoints(list);

list.addElement(a);
list.addElement(b);
list.insertElement(c,0);
printPoints(list);

list.removeElement(1);
printPoints(list);

list.setElement(0, new Point(9,9));
printPoints(list);
```

Stubs

- This gets the code compiling and running
- Eclipse generates these stubs for us

```
public class DynamicList {

    public int getSize() {
        // TODO Auto-generated method stub
        return 0;
    }

    public Point getElement(int x) {
        // TODO Auto-generated method stub
        return null;
    }

    public void addElement(Point a) {
        // TODO Auto-generated method stub
    }

    public void insertElement(Point c, int i) {
        // TODO Auto-generated method stub
    }

}
```

Based on an Array

- Maintain a private pointer to an array of Points
- Grow/shrink the array as needed

```
public class DynamicList {

    private Point [] data;

    public DynamicList() {
        data = new Point[0];
    }

    public int getSize() {
        return data.length;
    }

    public Point getElement(int x) {
        return data[x];
    }

    public void setElement(int x, Point p) {
        data[x] = p;
    }
}
```

Growing the Array

```
public class DynamicList {

    public void addElement(Point a) {
        // New array with an extra space
        Point [] tmp = new Point[data.length+1];

        // Copy old array to new
        for(int x=0;x<data.length;++x) {
            tmp[x] = data[x];
        }

        // Add new element to end
        tmp[data.length] = a;

        // Old array is now garbage
        data = tmp;
    }

}
```

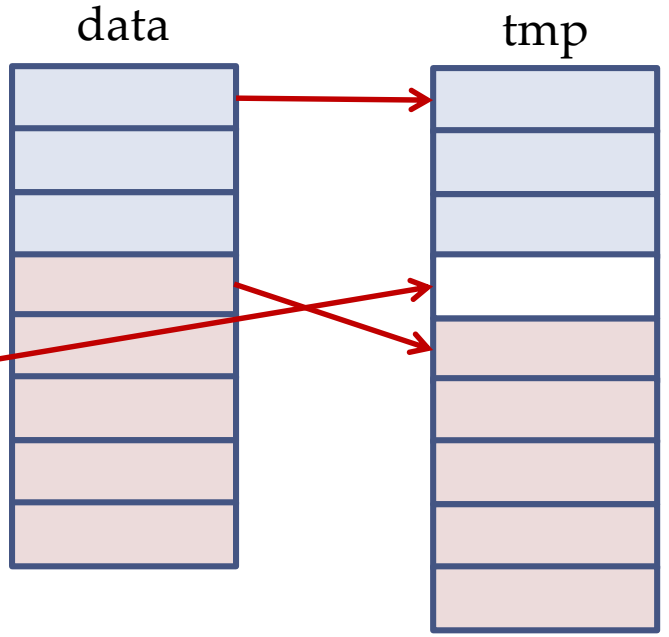
```
Point a = new Point(1,1);
Point b = new Point(2,2);
Point c = new Point(3,4);

DynamicList list = new DynamicList();
printPoints(list);

list.addElement(a);
list.addElement(b);
list.insertElement(c,1);
printPoints(list);
```

```
1,1
2,2
```

Array Copy

```
public void insertElement(Point c, int i) {
    // New array with an extra space
    Point [] tmp = new Point[data.length+1];

    // Copy entries before the new element
    //               src  srcPos  dst  destPos  count
    System.arraycopy(data, 0,     tmp, 0,       i);

    // Copy entries after the new element
    System.arraycopy(data, i, tmp, i+1, data.length-i);

    // Add the new element
    tmp[i] = c;

    // Old array is now garbage
    data = tmp;

}
```

i = 3




```
1,1
3,4
2,2
```

Left to the Student

- Write the "remove" code
- Better algorithm for growth
  - Capacity and Current-Size
  - Double each growth
- Constructor parameters
  - Initial size
  - Growth pattern

General Solution

- There is nothing "Point" specific in the list
- It really just collects addresses
- Use "Object" since it is the base class of all

```
public class DynamicList {

    private Object [] data;

    public DynamicList() {
        data = new Object [0];
    }

    public Object getElement(int x) {
        return data[x];
    }

    public void setElement(int x, Object p) {
        data[x] = p;
    }
```

```
    public void addElement( Object a) {
        // New array with an extra space
        Object tmp = new Object [data.length+1];

        // Copy old array to new
        for(int x=0;x<data.length;++x) {
            tmp[x] = data[x];
        }

        // Add new element to end
        tmp[data.length] = a;

        // Old array is now garbage
        data = tmp;
    }

}
```

Nasty Casts

- You can put any object in the list. We would like the compiler to catch mistakes.
- Catching mistakes at compile-time is much better than finding them at runtime.
- You have to cast when you get items from the list.

```
DynamicList list = new DynamicList();
list.addElement(new Point(1,2));

list.addElement(new Line()); // OOPS

Point p = (Point)list.getElement(1);
```

Generics

- You want the interface to be specific
- You can substitute these places with ANY class name

```
public class DynamicList {

    private Object [] data;

    public DynamicList() {
        data = new Object[0];
    }

    public int getSize() {
        return data.length;
    }

    public Point getElement(int x) {
        return (Point)data[x];
    }

    public void setElement(int x, Point p) {
        data[x] = p;
    }
```

```
    public void addElement(Point a) {
        // New array with an extra space
        Object [] tmp = new Object[data.length+1];

        // Copy old array to new
        for(int x=0;x<data.length;++x) {
            tmp[x] = data[x];
        }

        // Add new element to end
        tmp[data.length] = a;

        // Old array is now garbage
        data = tmp;
    }
```

Generics

- You can tell the compiler to do this substitution for you.
- The "angle brackets" contain the substituted value.

```
public class DynamicList<T> {

    private Object [] data;

    public DynamicList() {
        data = new Object[0];
    }

    public int getSize() {
        return data.length;
    }

    public T getElement(int x) {
        return (T)data[x];
    }

    public void setElement(int x, T p) {
        data[x] = p;
    }
```

```
DynamicList<Point> points = new DynamicList<Point>();
points.addElement(new Point(1,2));
Point p = points.getElement(0);

DynamicList<Line> lines = new DynamicList<Line>();
lines.addElement(new Line());
Line a = lines.getElement(0);
```

Generics in Action

- You may rarely define your own generics.
- But the java library uses them – especially the collections.
- The class we just developed is already in the library.
- You will definitely have to USE generics.
- Think of the "angle brackets" as another parameter used by the compiler.
- There can be multiple substitution parameters:

```
HashMap<String,Point> map = new HashMap<String,Point>();
```

Built-ins as Objects?

- How can we put built-in types like int and float in our dynamic list?
- We need a wrapper class to contain the built-in within an object:

```
public class Integer {

    private int value;

    public Integer(int val) {
        value = val;
    }

    public int getValue() {
        return value;
    }

}
```

```
DynamicList<Integer> ages = new DynamicList<Integer>();

int x = 10;
ages.addElement(new Integer(x));

ages.addElement(new Integer(20));

Integer q = ages.getElement(1);
int z = q.getValue();

z = ages.getElement(0).getValue();
```

Boxing

- This is so common that the java.lang has these built-in wrappers already defined
- Used to "box up" a built-in into an object
- Static methods to go to and from String values

```
Integer i = new Integer(20);
int x = i.intValue();

String g = Integer.toString(x);
String j = Integer.toString(x,16); // Hex

x = Integer.parseInt("1234");
x = Integer.parseInt("FACE",16);
```

- Boolean, Long, Double, Float, Short, Byte, Integer, Character

Automatic Boxing

- The compiler knows about these classes. The compiler will stick in the code for you.
- This is just a compiler trick. The resulting code is the same as if you spelled it out.

```
//ages.addElement(new Integer(245));
ages.addElement(245); // automatic "new Integer(245)"

//int x = ages.getElement(0).intValue();
int x = ages.getElement(0); // automatic call to "intValue"
```

Your Turn

- Code up the DynamicList for Point (hardcode it for Points).
- Add the "remove" method.
- Write test code to test inserting and removing from the ends of the list and from the middle.
- Have a look at java.util.ArrayList. How are the method names different? What other methods did we leave out? (We'll look at ArrayList in depth next time.)