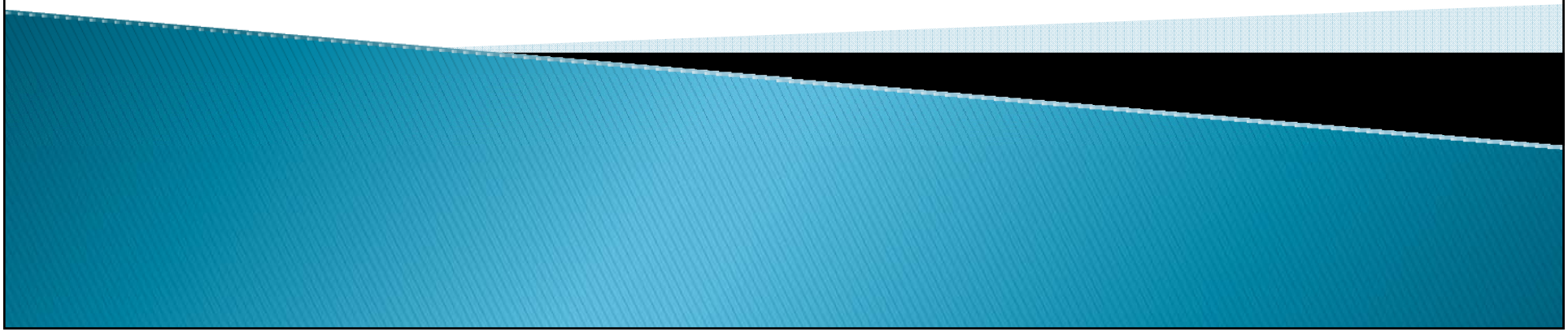
CSSE 220 Day 11

Generic types
Paint design



Announcements

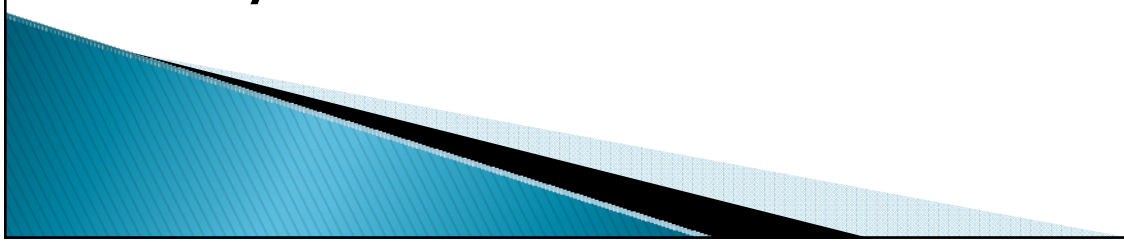
- Please sit with your Paint project partner
- Solutions to HW6 written problems and DotsUML should be in the usual place on ANGEL:
  Lessons > Assignments > Solutions
- Today:
  - Statics revisited
  - Compositing
  - Generic types in Java.
  - Meet your Paint partner, finish your UML diagram and work on your IEP, both due Friday. 5pm.
- Questions on BallWorlds? Exam?

Static

- Please answer quiz questions 1 and 2
- Demo
- Do Quiz questions 3 and 4

Returning Multiple Values From a Method

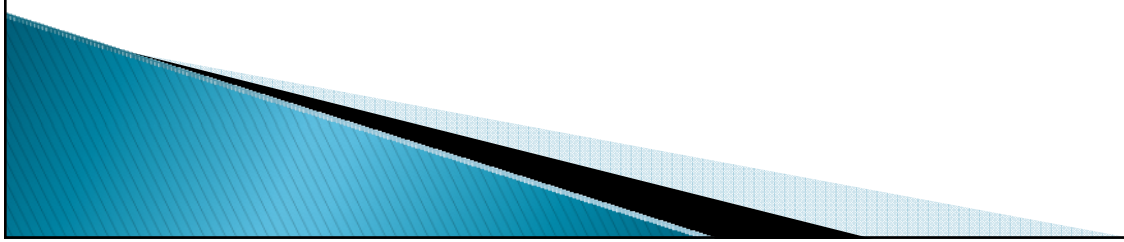
- In Python we could simply write
  return x, y
- In C, we could pass pointers to variables and change what they pointed to.
- What can we do in Java?
- This is a simple example of what is called the Composite Pattern.
  The returned value is a composition of two or more values that may be unrelated other than by the need to be returned from a function



Generics

- We really want our algorithms to operate on any type of data, without having to re-write the whole method.

- In Java, we can do this two ways:
  - Use inheritance (pre-Java 1.5, a bit clunky)
  - Use *Generics* (newer, nicer)



Using Inheritance in Java 1.4

```
ArrayList list = new ArrayList();
list.add(new Integer(3)); // 3 needs to be boxed
list.add("hello");
Integer temp = (Integer)list.get(0); //casting
int num = temp.intValue(); // unboxing
//int num = list.get(0); // I wish this worked!
```

Problems?

- Casting, boxing and unboxing are a pain in the neck!
- We have no control over the type of what goes in! (which means we should check for compatibility using instanceof to avoid ClassCastExceptions)

Using Inheritance in Java 1.5

```
ArrayList list = new ArrayList();
list.add(3); //auto-boxed to an Integer
list.add("hello");
int num = (Integer)list.get(0); // auto-unboxed
int num2 = (Integer)list.get(1) // Class-cast exception!
//int num = list.get(0); // still doesn't work
```

Problems?

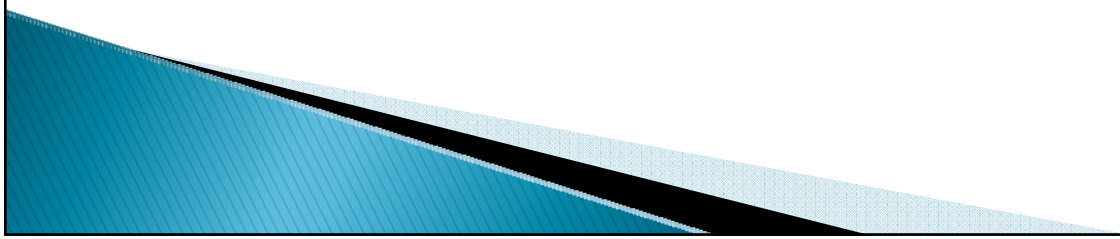
- Casting is still a pain in the neck!
  - At least auto-boxing relieves some of the pain!
- We still have no control over the type of what goes in!

Using Generics in Java 1.5

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(3);
//list.add("hello"); // now a compile-time error
int num = list.get(0); // I'm happy this works!
```

Problems?

- Casting? Not needed!
- Mixed types? Caught at compile time!



Creating code with Generics

- To use generics, use the type in a parameter.
- Example showing:
  - The use of a type in a class
  - Various places where the type parameter can be used:

```
public class SomeClass<E> {
   public E someMethod(E param) {
     E retValue = 2 * param;
     return retValue;
   }
   …
}
```

- Unfortunately, this example doesn't work, since we can't multiply 2 by an unknown, possibly non-numeric type.

- Do LeechHome quiz question

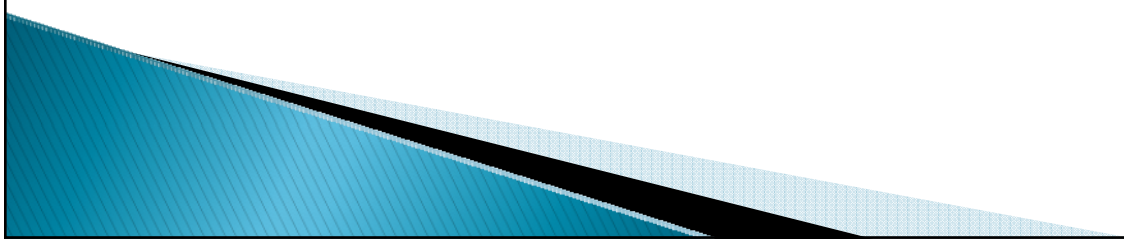
Going further

- What if I have a method that operates on an ArrayList of Vehicles, but I want to pass an ArrayList of Trucks?
- Intuitively, this should work, but Java doesn't allow it, since it couldn't catch errors until runtime.
- Solution? In the method declaration, use type bounds with wildcards:
- public void processVehicle(ArrayList<? extends Vehicle> list) {
  - for (Vehicle v : list) { … }

  }

Type erasure

- At compile time, the generics are replaced with the types used
- If there are bounds, it uses them and inserts the proper casts

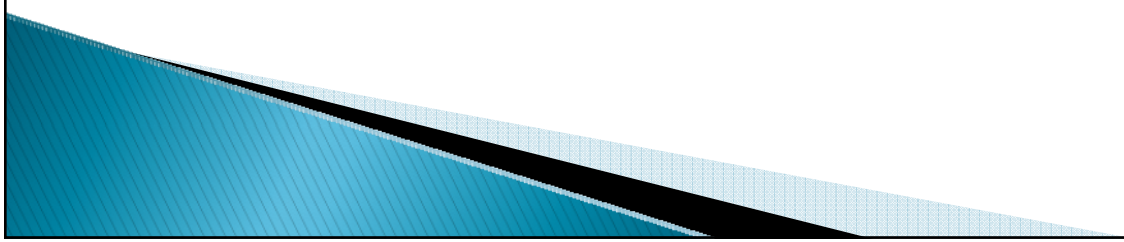


Some Limitations

- Can't use primitives as types
  - No int, need to use Integer
- Can't instantiate a type: E foo = new E();
  - What is E? It could even be an abstract class; this wouldn't make sense!
- Can't make generic arrays: E[] ar = new E[17];
  - Naïve solution: use typecasts:
    - E[] ar = (E[])(new Object[17])
    - This gives a compiler warning
  - Better solution: use ArrayList<E>

How could generics have helped BigRational?

- Check out the demo.



java.lang

Interface Comparable<T>

Type Parameters:

T - the type of objects that this object may be compared to

- Any class that implements Comparable contracts to provide a **compareTo** method.

Method Detail

String is a Comparable class.
If it did not already have a compareTo method, how would you write it?

compareTo

```
int compareTo(T o)
```

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

- Therefore, we can write generic methods on Comparable objects. For example, in the Arrays class:

static void	sort(Object[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of objects into ascending order, according to the natural ordering of its elements.

Example of using Arrays.sort

```
import java.util.Arrays;

public class StringSort {

   public static void main(String[] args) {
      String [] toons = {"Mickey", "Minnie", "Donald",
                         "Pluto", "Goofy"};
      Arrays.sort(toons);
      for (String s:toons)
         System.out.println(s);
   }
}
```

Output:

```
Donald
Goofy
Mickey
Minnie
Pluto
```

Project Time