

Announcements/Follow-ups

- P6 posted, due next Wednesday
- Quiz tomorrow on Collections and generics
- Follow-ups:
 - Enums
 - Bidirectional maps
 - getClass debug
- Finish collections:
 - Concurrent modification
 - Iterators
 - Collections Example: temporary storage

Generic types

Analogy with methods:

- *method(argument)*
 - Methods can be supplied with arguments that affect their behavior. The argument itself is something with a value:
 - A literal
 - A primitive variable
 - An object
 - An expression
- *generic_type<type_argument>*
 - Generic types are supplied with arguments that affect their behavior. The argument itself is a data-type:
 - A class
 - An enum
 - An interface
 - (not a primitive)

Generic Usage

- Collections: “One-size-fits-all”

```
ArrayList<String> strs = new ArrayList<String>();  
ArrayList<Integer> ints = new ArrayList<Integer>();
```

Vs

```
Object[] objs = new Object[0];
```

- ArrayList class is only defined once
- No repetitive code for StringList, IntegerList, etc.
- Type safety:
 - No casting to and from object
 - Compiler-time warnings for bad type-casting

Generic Usage

- The comparable interface:
- <http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html>
- Implemented by
 - String
 - Integer
 - Date
 - File (?)
 - Many more
- Comparable example

Lexicographical ordering

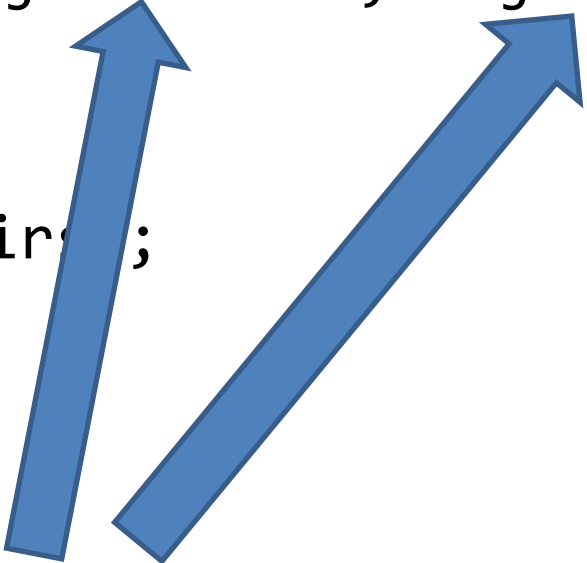
- The mathematical generalization of alphabetical ordering:
 - Compare “Strings” from left to right
 - At the first mismatch, the “lower” letter determines the “lower” “String”
 - For example:
 - “aaab” < “aaac” < “abaa”
 - [0,0,0,1] < [0,0,0,2] < [0,1,0,0]

Generic methods

```
static void method(Object first, Object second)
{
    //Do stuff
    Object temp = first;
    //Do more stuff
}
```

Generic methods

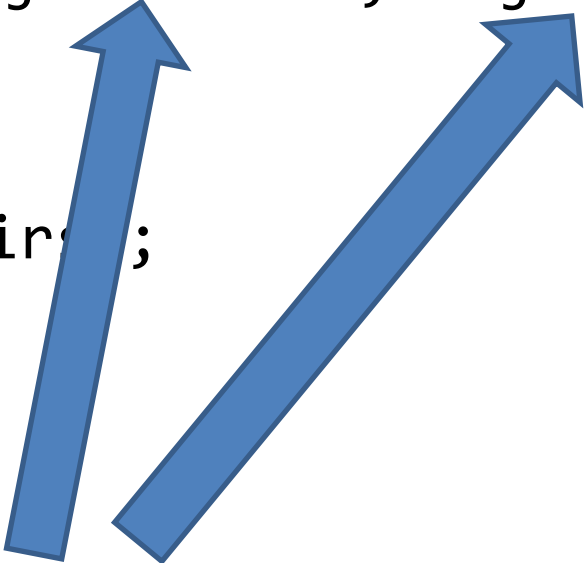
```
static void method(Object first, Object second)
{
    //Do stuff
    Object temp = first;
    //Do more stuff
}
```



“Arguments” or “Parameters”?

Generic methods

```
static void method(Object first, Object second)
{
    //Do stuff
    Object temp = first;
    //Do more stuff
}
```



Parameters.

Arguments: when it's called

Parameters: when it's defined

Generic methods

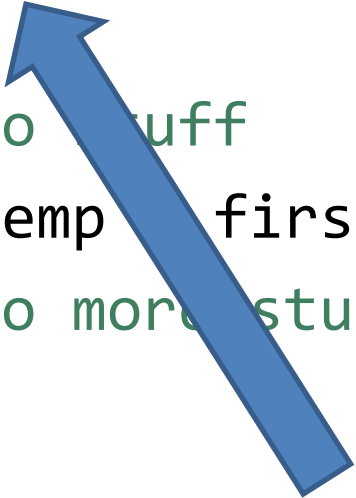
```
static void method(Object first, Object second)
{
    //Do stuff
    Object temp = first;
    //Do more stuff
}
```



Value is copied to
every occurrence
in the method
body

Generic methods

```
static <T> void method(T first, T second)
{
    //Do stuff
    T temp = first;
    //Do more stuff
}
```



“Type Parameter”

Single upper-case letter is
customary (NON-descriptive...)

Generic methods

```
static <T> void method(T first, T second)
{
    //Do stuff
    T temp = first;
    //Do more stuff
}
```

The diagram illustrates the concept of a generic type parameter. It shows a code snippet with a generic method. Three blue arrows originate from the '<T>' in the signature: one points to the '<T>' in the parameter list, another points to the '<T>' in the variable declaration 'T temp', and a third points to the '<T>' in the parameter 'T second'. This visualizes how the same type 'T' is used throughout the method, representing a single abstract type.

“Value” is copied to every occurrence in the method body

- Not really a “Value”, but a data-type.
- “Meta”

Generic methods

```
Meta(T) {  
    static <T> void method(T first, T second)  
    {  
        //Do stuff  
        T temp = first;  
        //Do more stuff  
    }  
}
```

The diagram illustrates the use of the generic type `T` within the `Meta(T)` class. Three blue arrows originate from the `T` in the class signature `Meta(T)` and point to the following uses of `T`:

- The first arrow points to the generic type parameter `<T>` in the method signature `static <T> void method(T first, T second)`.
- The second arrow points to the parameter type `T` of the `first` argument in the method signature.
- The third arrow points to the parameter type `T` of the `second` argument in the method signature.

- Generic Swap Example

Generic Interfaces

- Comparable: Intended for any class whose objects can be ordered.
- Interface signature:
 - `public interface Comparable<T>`
- Single method signature:
 - `int compareTo(T o)` //access modifier?
- MyInteger vs. MyComparableInteger

Type Erasure

- Really “Type Parameter Erasure”
- Occurs during compilation
- Resulting byte-code consists of ordinary (non-generic) classes, interfaces, and methods.
 - Type-parameters are replaced by Object, or more specific types when possible
 - All necessary type-casts are inserted
 - Effectively, it translates generic swap to non-generic swap automatically
 - Essentially no run-time cost
- BidirectionalMapExample revisited

Collection algorithms

- There is a class called “Collections” with many static methods for manipulating collections
 - Search, sort, shuffle, reverse, rotate, fill, etc.
 - All generic
 - Contrast with Arrays class: all overloaded
- Collections example