Generics

Week 3 Presentation 1

Recursion is very important with complex collections of objects

RECURSION is very useful for processing structured COLLECTIONS of OBJECTS

As we have seen with quick-sort: inside an array you can find smaller sub-arrays. Inside a set you find subsets. And when you divide enough, you get empty or single element sets.

GENERICS in JAVA

Before we start to discuss collections we will talk about "Generics" which are very important for reusing code in Java. You probably have already seen ArrayList, this is an example of generics.

Strong Typing and Weak Typing

Strong Typing

- It means that the compiler will NOT let you mix variables of different types, unless they are known to be compatible (such as int and float)
- It protects against unwanted effects
- Java is strongly typed

Weak Typing

- Many scripting languages take the opposite approach and guess the type of variables from how you are using them
- Perl is an example of a weak typing


I LIKE THE STRONG SILENT TYPE

Java is strong typed, But...

```
4
5       String[] months = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
6               "Sep", "Oct", "Nov", "Dec"};
7
8       int[] sales = {120, 98, 75, 110, 150,
9               180, 170, 174};
10
```

```
Jan     Feb     Mar     Apr     May     Jun     Jul     Aug     Sep     Oct     Nov     Dec
120     98      75      110     150     180     170     174
```

The problem with strong typing is that for instance if you want to write a methods that displays on a line elements from an array, the same method cannot handle an array of Strings and an array of ints. It's one or the other, exclusively.

Java is strong typed, But…

```
15
16      public static void displayArr(String[] arr) {
17          int n = arr.length;
18          for (int i = 0; i < n; i++) {
19              if (i > 0) {
20                  System.out.print("\t");
21              }
22              System.out.print(arr[i]);
23          }
24          System.out.println("");
25      }
```

- If this is the function that displays an array of Strings …

```java
    public static void displayArr(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n; i++) {
            if (i > 0) {
                System.out.print("\t");
            }
            System.out.print(arr[i]);
        }
        System.out.println("");
```

Java is strong typed, But…

- … you must overload it to display an array of integers. Note that, apart from the parameter type
- The code is identical except for the parameter
- Called overloading

Overloading

- Overloading allows you to give the same name to several functions. By looking at parameters, Java will **know** which one to call
- Different versions of a function
  - Same name
  - Same return type
  - Different parameters

Method *Signature*

- The linker of the class loader that will match your code to methods with the same name that are available
- What defines the "signature" of a function?
  - The number of parameters
  - And their types

```
substring(int beginIndex)
Returns a new string that is a substring of this

substring(int beginIndex, int endI
Returns a new string that is a substring of this
```

```java
public class Overloading {

static int function(int n) {
    System.out.println("This is function(int)");
    return 0;
}
static int function(double x) {
    System.out.println("This is function(double)");
    return 0;
}

    public static void main(String[] args) {

        int n = 1
        double va
        float f =

        n = funct
        n = function(val);
        n = function(f);
    }
}
```

```
This is function(int)
This is function(double)
This is function(double)
```

In this example, when we call the function with a float, it's automatically "upgraded" to double and the function for doubles is called (the same function would be called for the int if there were no special function for integers)

Waste of time.
Change: must be repeated in all methods.

- If overloading helps keep the code safe, it's a waste of time to write identical code several times (even if we copy and paste). Worse, if we want to change the code, for instance to separate array elements by semicolons (;) instead of tabs when we print them, we must modify every method.

Code that works
With Strings and integers!?

IMPORTANT: Generics only works with OBJECTS

BEWARE: in Java, generics ONLY WORK with objects (references). Base variables, such as int, float, char, boolean variables ARE NOT objects. However, all base types have a "shadow" corresponding class (Integer, Float, Character, Boolean ...). For using generics, we must use these classes, which convert automatically to and from base data types (operations known as "boxing" and "unboxing")

```
4
5       String[] months = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
6               "Sep", "Oct", "Nov", "Dec"};
7
8       int[] sales = {120, 98, 75, 110, 150,
9               180, 170, 174};
10
```

```
Jan     Feb     Mar     Apr     May     Jun     Jul     Aug     Sep     Oct     Nov     Dec
120     98      75      110     150     180     170     174
```

- If we have an array of Strings (which are objects) and an array of Integers (that are objects too), then we can create a single method that works for both, without any explicit overloading.

GENERICS:
Computer-aided Overloading

- "Generics" comes from a Latin word that means "family" or "kind". It can be seen as automatic overloading.
- Once again (worth repeating) it only works with object references in Java.

Generic Display

```
3
4
5       String[] months = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
6                          "Sep", "Oct", "Nov", "Dec"};
7
8       Integer[] sales = {120, 98, 75, 110, 150,180, 170, 174};
9
```

```
36
37      static <T> void displayArray(T[] arr) {
38          int n = arr.length;
39
40          for (int i = 0; i < n; i++) {
41              if (i > 0) {
42              System.out.print("\t");
43              }
44              System.out.print(arr[i]);
45              }
46          System.out.println("");
47      }
48
49      public static void main(String args []) {
50          GenericDemo d = new GenericDemo();
51          displayArray(d.months) ;
52          displayArray(d.sales) ;
53      }
54  }
55
56
```

- Before the return type of the method, you specify one (or several) symbols between angular brackets that represent Classes.
- Objects

You need to specify the symbol before the return type because the return type of the generic method could perfectly be defined as T. If you need several generic classes (for instance one for the return type and one for the parameter, or because you want to pass parameters from two different classes), you separate them with commas.

<T,U>

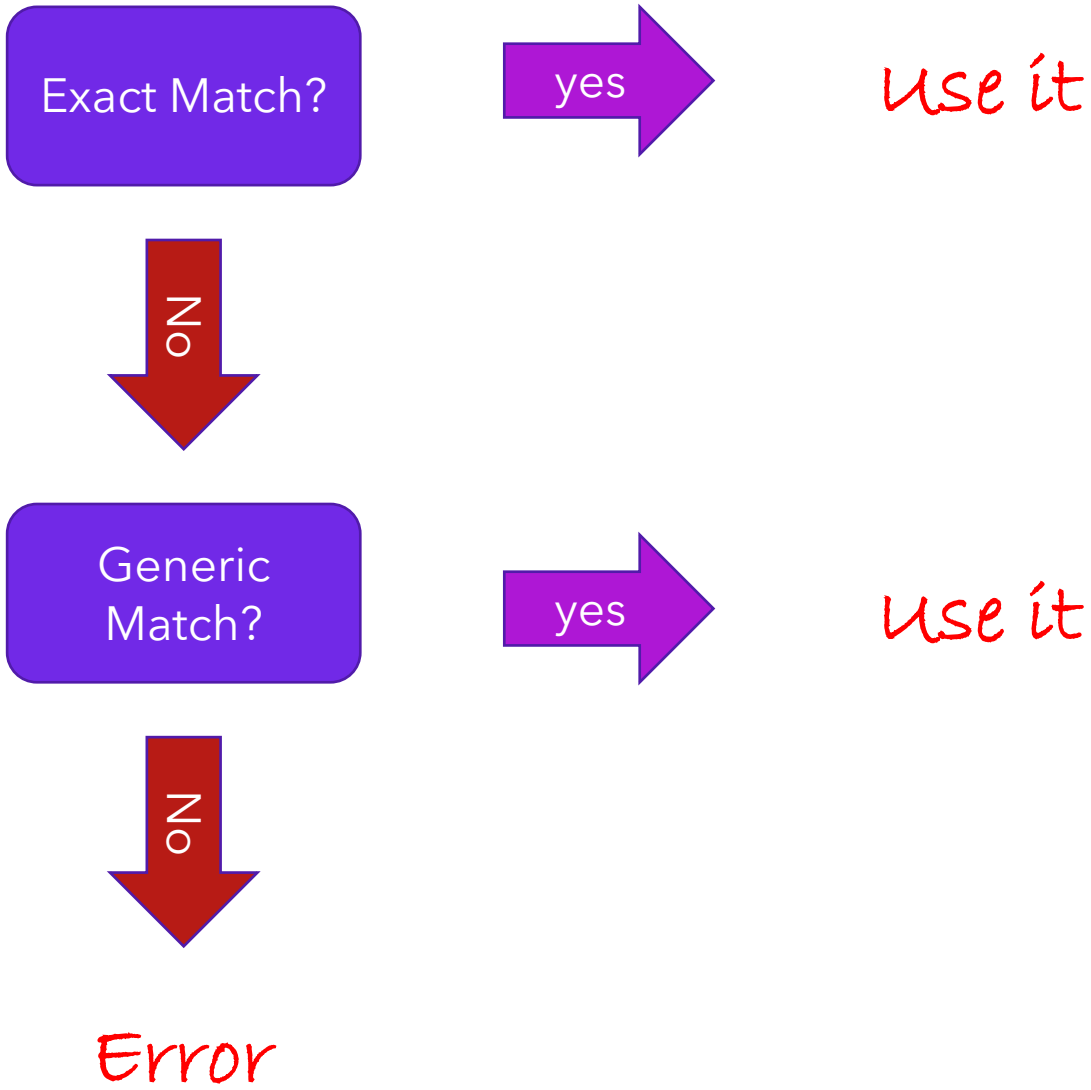
Generic methods can be overloaded

```
public static <T> void displayArray(T[] arr);

public static <T> void displayArray(T[] arr, String sep);

public static void displayArray(Date[] arr, String format);
```

Like any method, generic methods can be overloaded, either by another generic method, or for instance to handle a very special case.



The linker in the Class Loader Subsystem will try to find the best match.

Functions can be generic.
Classes can also be generic.

Important for collections!

This idea of turning a class into a kind of parameter can also be applied to classes; this is important for collections, which are "container classes". For instance, an ArrayList is a class that can hold (contain) objects of any class.

Class ArrayList<E>

java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractList<E>
            java.util.ArrayList<E>

```java
        public void add(T t) {
            this.t = t;
        }

        public T get() {
            return t;
        }

        public static void main(String[] args) {
                Box<Integer> integerBox = new Box<Integer>();
                Box<String> stringBox = new Box<String>();

                integerBox.add(new Integer(10));
                stringBox.add(new String("Hello World"));

                System.out.printf("Integer Value :%d\n\n", integerBox.get());
                System.out.printf("String Value :%s\n", stringBox.get());
            }
}
```

```
<terminated> Box [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/Home/bin/ja
Integer Value :10

String Value :Hello World
```

<terminated> Box [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/Home/bin/java (27 Feb 2020, 2:00:30 pm)