Arrays & Strings

Arrays of Native Type Values

- A group of native values.

  Each native value has an index (starting at 0)

  Represented as an object.

- Creating the array of native type values has two stages:

  Declaring the variable which that holds the object reference (The array in Java is an object).

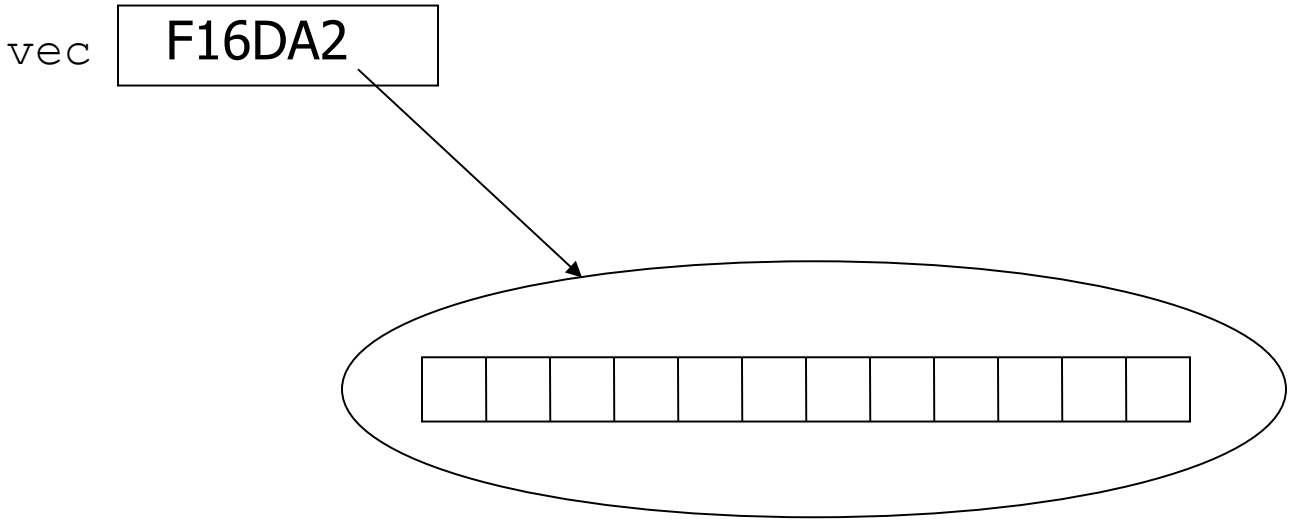
  ```
  int vec[];
  ```

  Instantiating the object

  ```
  vec = new int[12];
  ```

Arrays of Native Type Values

- The result of these two statements is:



Arrays of Native Type Values

- Now it is possible to work with the created array similarly to C\C++.

```
vec[0] = 12;
Vec[2] = vec[0] + 3;
```

Arrays of Objects' References

- A group of objects' references.

  Each reference has an index (starting at 0). The whole group is represented using an object. The array is an object.

- Creating the array has three stages:

  Declaring the variable that holds the object's reference (the array in Java is an object).

  ```
  Student vec[];
  ```

  Creating the object (creating the array).

  ```
  vec = new Student[12];
  ```

Arrays of Objects' References

- When the array is created it holds in its cells `null` values.
- Instantiating the objects and place their references within the cells is the third stage. It isn't a must. We can wait with that.

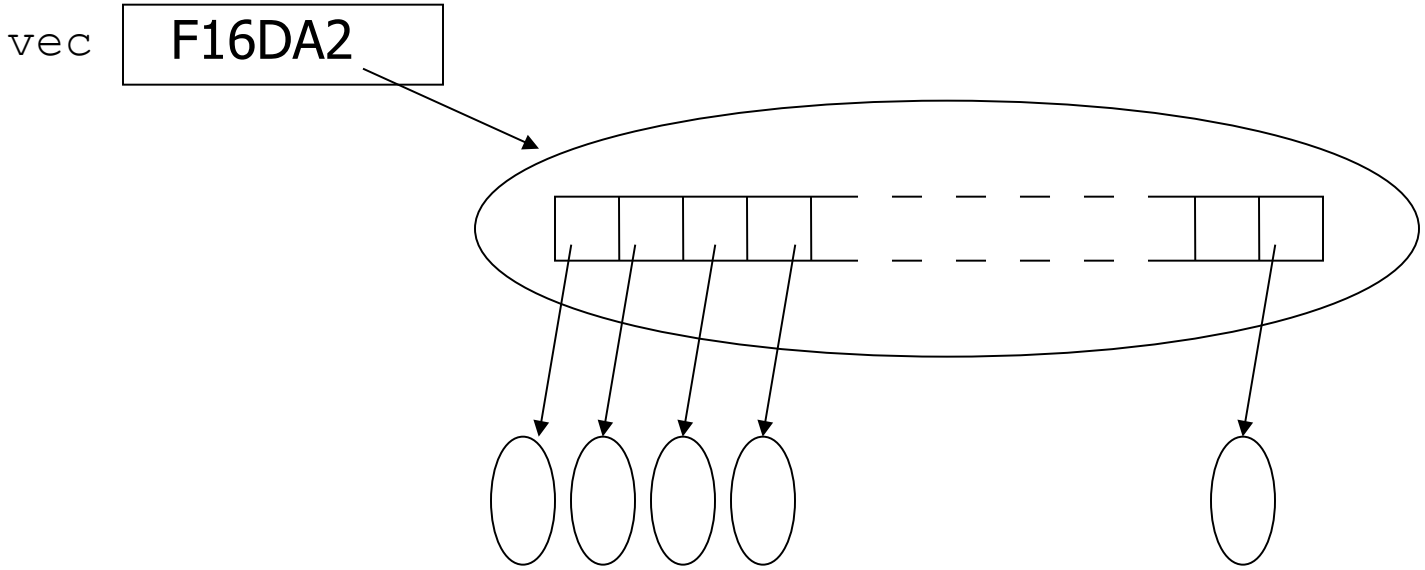
Arrays of Objects' References

- If we want to have all objects created at once we can use a simple loop:

```
for(int i=0; i<12; i++)
{
     vec[i] = new Student();
}
```

Arrays of Objects' References

- The result of these three statements is:



Arrays of Objects' References

- Now it is possible to work with the created array similarly to C++.

```
vec[0] = new Student ("Haim");
Vec[2] = vec[0];
```

Arrays of Objects' References

The Detailed Syntax

```
Student vec[];
vec = new Student[3];
vec[0] = new Student("Moshe");
vec[1] = new Student("David");
vec[2] = new Student("Ramy");
```

The Short Syntax

```
Student vec[] = {   new Student("Moshe"),
                    new Student("David"),
                    new Student("Ramy")     };
```

The Square Brackets Position

- The square brackets can be placed either before the variable name or after it.
- Placing the square brackets before or after has a different meaning.

```
int vec[], number1, number2;
```

— number1 and number2 are simple variables

```
int []vec, number1, number2;
```

— number1 and number2 are variables that can hold a reference for array

Copying Array Values

- In order to copy the values of one array to an other one you should use the method `System.arraycopy()`

```
public static void arraycopy(
                Object src,
                int src_position,
                Object dst,
                int dst_position,
                int length                    )
```

Multi-Dimensional Array

- A multi-dimensional array is an array of arrays.
- There are two ways for creating multi-dimensional arrays:

Detailed Way

```
int matrix[][];
matrix = new int[3][];
matrix[0] = new int[4];
matrix[1] = new int[4];
matrix[2] = new int[4];
```

Short Way

```
int matrix[][] = new int[3][4];
```

The length variable

- Each array has a variable named length.
- The length variable holds the size of the array.

```
int vec[] = {12,32,42,55};
for(int i=0; i<vec.length; i++)
{
    System.out.println(vec[i]);
}

int mat[][] = {{1,2,3,7,8}, {3,2}, {4,6,5,5}};
```

The For Each Loop

- When there is a need to go over items contained within a collection or an array.

```
for( typeName ___ : ___ )
{
    ...
    ...
}
```

The collection \ array we want to iterate

The element in each iteration

The general type for all elements

The For Each Loop

```
.
.
.
public double getTotal(double []numbers)
{
 double total=0;
 for(double num : numbers)
 {
     total += num;
 }
 return total;
}
.
.
.
```

The For Each Loop

```
public class ForEachSample
{

    public static void main(String args[])
    {
            Rectangle[] vec = {
                    new Rectangle(3,4),
                    new Rectangle(5,6),
                    new Rectangle(6,2),
                    new Rectangle(8,3)
            };

            for(Rectangle rec : vec) {
                System.out.println(rec);
            }
    }
}
```

The For Each Loop

```
/Applications/jdk1.8.0/bin/java -Didea.launcher.port=7533 "-Didea.launcher.bin.path=/Appl
rectangle width=3.0 height=4.0
rectangle width=5.0 height=6.0
rectangle width=6.0 height=2.0
rectangle width=8.0 height=3.0

Process finished with exit code 0
```

The For Each Loop

```
.
.
.
public double getTotal(Collection<Double> numbers)
{
 double total=0;
 for(Double num : numbers)
 {
     total += num.doubleValue();
 }
 return total;
}
.
.
.
```

The For Each Loop

```
public class ForEachSample
{

    public static void main(String args[])
    {
        ArrayList<Rectangle> array = new ArrayList<Rectangle>();
        array.add(new Rectangle(3,4));
        array.add(new Rectangle(5,6));
        array.add(new Rectangle(6,2));
        array.add(new Rectangle(8,3));
        for(Rectangle rec : array) {
            System.out.println(rec);
        }
    }
}
```

The For Each Loop

```
/Applications/jdk1.8.0/bin/java -Didea.launcher.port=7533 "-Didea.launcher.bin.path=/Appl
rectangle width=3.0 height=4.0
rectangle width=5.0 height=6.0
rectangle width=6.0 height=2.0
rectangle width=8.0 height=3.0

Process finished with exit code 0
```

The `String` Class

- Each text is represented by a String object.
- The string object holds all the characters of the text it represents.

  Each character is represented using its unicode value. There is no null value to indicate the end of the string
- Once the String class was instantiated it isn't possible to change the text it represents.

Instantiating the `String` class

- Two common ways of getting a String object.

```
String str = "abc";
String str = new String("abc");
```

- String has various constructors:

```
public String(String str)
public String(byte vec[])
public String(char vec[])
public String(byte vec[], String encode)
```

Comparing strings

- Comparing two string using the equal sign isn't correct.

```
String str1 = new String("Haim");
String str2 = new String("Haim");
if(str1==str2)
     ...
```

- Comparing two strings should be done using equals().

```
String str1 = new String("Haim");
String str2 = new String("Haim");
if(str1.equals(str2))
     ...
```

The `toString()` method

- Each object in Java is also an `Object`. The `toString()` method is defined within the `Object` class.

  The toString() version in class Object returns a string that consists from the name of the class and from the value that the hashcode() method returns when invoked on the given object.

  Each time, an `Object` reference is passed over to the print()\println() methods the text printed on screen is the text the `toString()` method returns.

The `toString()` Method

- The `toString()` method can be overridden in every new class we define.
- The following example presents the use of the `toString()` method.

The `toString()` Method

```
public class ToStringDemo
{
      public static void main(String args[])
      {
         Puppy pupic = new Puppy("Shimshon", 3);
         Student stud = new Student("Chris", 19);
         System.out.println(pupic);
         System.out.println(stud);
      }
}
```

The `toString()` Method

```
class Puppy
{
 private int age;
 String name;
 Puppy(String nameVal, int ageVal)
 {
     name = nameVal;
     age = ageVal;
 }
```

The `toString()` Method

```
public String toString()
 {
     return "[name="+name+",age="+age+"]";
 }
}
```

The `toString()` Method

```
class Student
{
 private int age;
 String name;
 Student(String nameVal, int ageVal)
 {
     name = nameVal;
     age = ageVal;
 }
}
```

The `StringBuffer` Class

- Unlike the String class, Instance of the `StringBuffer` class represents a changeable text.

```
StringBuffer sb = new StringBuffer("haim");
sb.append(" ");
sb.append("michael");
```

The `StringBuffer` Class

- When adding one string to another, sometimes, especially in loops, using the `StringBuffer` class might be more efficient.

```
StringBuffer sb = new StringBuffer();
String vec[];
...
for(int i=0; i<vec.length; i++)
{
        sb.append(vec[i]);
}
```

Using The `StringBuffer` Class

The `StringBuffer` Class

```
String str = new String("");
String vec[];
...
for(int i=0; i<vec.length; i++)
{
     str = str + vec[i];
}
```

Without Using The `StringBuffer` Class

The `StringBuffer` Class

```
package com.lifemichael.samples;

public class BadCode {

    public static void main(String[] args) {
        String str = "";
        String strings[] =
            {"a","b","c","d","e","f","g","h","i"};
        long number = System.currentTimeMillis();
```

The `StringBuffer` Class

```
        for(int i=0; i<20000; i++)
        {
            if(i%1000==0)
            {
                System.out.println(i);
            }
            for(int k=0; k<strings.length; k++)
            {
                str = str + i + strings[k];
                // str = (new StringBuffer(str).append(i).
                //         append(strings[k])).toString()
            }
        }
        long result = System.currentTimeMillis() - number;
        System.out.println(result);
    }
}
```

The `StringBuffer` Class

```
package com.lifemichael.samples;

public class GoodCode {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();
        String strings[] =
            {"a","b","c","d","e","f","g","h","i"};
        long number = System.currentTimeMillis();
```

The `StringBuffer` Class

```
        for(int i=0; i<20000; i++)
        {
            if(i%1000==0)
            {
                System.out.println(i);
            }
            for(int k=0; k<strings.length; k++)
            {
                sb.append(i).append(strings[k]);
            }
        }
        long result = System.currentTimeMillis() - number;
        System.out.println(result);
    }
}
```

The `StringBuffer` Class

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help

Java - stringbufferdemo/src/com/lifemichael/samples/BadCode.java - Eclipse - /Users/haimmichael/Documents/java_ee_workspace_new

Package Expl

- hellojersey
- samplejsp
- Servers
- stringbufferdemo
  - src
    - com.lifemichael.sam
      - BadCode.java
      - GoodCode.java
  - JRE System Library [Java

BadCode.java

```
12          for(int i=0; i<20000; i++)
13          {
14              if(i%1000==0)
15              {
16                  System.out.println(i);
17              }
18              for(int k=0; k<strings.length; k++)
19              {
20                  str = str + i + strings[k];
21                  // str = (new StringBuffer(str).append(i).append(string
22              }
23          }
24          long result = System.currentTimeMillis() - number;
25          System.out.println(result);
```

Problems Javadoc Declaration Console

<terminated> BadCode [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0.jdk/Contents/Home/bin/java (Sep 17, 2014, 4:40:22 PM)

```
19000
254425
```

com.lifemichael.samples.BadCode.java - stringbufferdemo/src

The `StringBuffer` Class

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help

Java – stringbufferdemo/src/com/lifemichael/samples/GoodCode.java – Eclipse – /Users/haimmichael/Documents/java_ee_workspace_new

Package Expl

- hellojersey
- samplejsp
- Servers
- stringbufferdemo
  - src
    - com.lifemichael.sam
      - BadCode.java
      - GoodCode.java
  - JRE System Library [Java

GoodCode.java

```
7        long number = System.currentTimeMillis();
8        for(int i=0; i<20000; i++)
9        {
10           if(i%1000==0)
11           {
12               System.out.println(i);
13           }
14           for(int k=0; k<strings.length; k++)
15           {
16               sb.append(i).append(strings[k]);
17           }
18       }
19       long result = System.currentTimeMillis() - number;
20       System.out.println(result);
21   }
```

Problems Javadoc Declaration Console

<terminated> GoodCode [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0.jdk/Contents/Home/bin/java (Sep 17, 2014, 4:28:41 PM)

```
19000
54
```

Writable Smart Insert 14 : 43

The `StringBuilder` Class

- The `StringBuilder` functions similarly to the `StringBuffer`.
- Unlike the `StringBuffer` class, `StringBuilder` is not thread safe. The `StringBuilder` doesn't have any internal implementation that protects it from synchronization problems.
- The `StringBuilder` implementation is much faster.

The `StringTokenizer` Class

- The `StringTokenizer` class enables taking a long string and split it to tokens, new little strings, according to the separator string we choose (e.g. “,”).
- The default separator string is “ “.

```
StringTokenizer st =
     new StringTokenizer(“Hais hobby is JAVA");
while (st.hasMoreTokens())
{
     System.out.println(st.nextToken());
}
```

The `StringTokenizer` Class

- The output of this code will be:

  Haim
  hobby
  is
  JAVA

The `StringTokenizer` Class

```
...
String str = "Haim : loves : JAVA : and JINI 2";
StringTokenizer st1 = new StringTokenizer(str,":",false);
while (st1.hasMoreTokens())
{
     System.out.println(st1.nextToken());
}
...
```

The `main` Method Arguments

- When executing a java application it is possible to pass over arguments to the main method. Each argument will be represented by a `String` object.
- The `args[]` parameter receives a reference to array that holds all references for these `String` objects.

Sending Arguments to `main`

- The following example presents passing parameters to the `main` method.

```
...
public static void main(String args[])
{
    for(int i=0; i<args.length; i++)
    {
        System.out.println(args[i]);
    }
}
...
```

Notes:

Any attempt to access an array element outside the bounds causes a runtime exception.

Notes:

Please note that when having an object it is possible to have its references placed in more than one variable (or an array place).

```
Studetn std = new Student();
Student vec[];
vec = new Student[12];
vec[4] = std;
```

Each change through std will effect the array, vec.

Since the array is an object, it is possible having an array and have its reference placed in two different variables:

```
Student vec1[], vec2[];
vec1 = new Student[12];
vec1[0] = new Student("Moshe");
vec1[2] = new Student("David");
.
.
vec2 = vec1;
```

Notes:

Each array, as it is created, has a default value in each of its cells. The default value in “native type array” is according to its type. The default value in “class type array” is null.

Notes:

Note that in case of array of objects' references, the arraycopy method copies the references themselves (these are the values in a class-type array).

Notes:

Note the possibility to create an array that doesn't have the same number of elements in each one of its lines.

Notes:

Note that mat.length equals to 3 and mat[0].length equals to 5.

The For Each Loop

- When there is a need to go over items contained within a collection or an array.

```
for( typeName ___ : ___ )
{
    ...
    ...
}
```

The collection \ array we want to iterate

The element in each iteration

The general type for all elements

The For Each Loop

```
.
.
.
public double getTotal(double []numbers)
{
 double total=0;
 for(double num : numbers)
 {
     total += num;
 }
 return total;
}
.
.
.
```

The For Each Loop

```
public class ForEachSample
{

    public static void main(String args[])
    {
            Rectangle[] vec = {
                    new Rectangle(3,4),
                    new Rectangle(5,6),
                    new Rectangle(6,2),
                    new Rectangle(8,3)
            };

            for(Rectangle rec : vec) {
                System.out.println(rec);
            }
    }
}
```


The For Each Loop

```
/Applications/jdk1.8.0/bin/java -Didea.launcher.port=7533 "-Didea.launcher.bin.path=/Appl
rectangle width=3.0 height=4.0
rectangle width=5.0 height=6.0
rectangle width=6.0 height=2.0
rectangle width=8.0 height=3.0

Process finished with exit code 0
```


The For Each Loop

```
.
.
.
public double getTotal(Collection<Double> numbers)
{
 double total=0;
 for(Double num : numbers)
 {
     total += num.doubleValue();
 }
 return total;
}
.
.
.
```

The For Each Loop

```
public class ForEachSample
{

    public static void main(String args[])
    {
        ArrayList<Rectangle> array = new ArrayList<Rectangle>();
        array.add(new Rectangle(3,4));
        array.add(new Rectangle(5,6));
        array.add(new Rectangle(6,2));
        array.add(new Rectangle(8,3));
        for(Rectangle rec : array) {
            System.out.println(rec);
        }
    }
}
```


The For Each Loop

```
/Applications/jdk1.8.0/bin/java -Didea.launcher.port=7533 "-Didea.launcher.bin.path=/Appl
rectangle width=3.0 height=4.0
rectangle width=5.0 height=6.0
rectangle width=6.0 height=2.0
rectangle width=8.0 height=3.0

Process finished with exit code 0
```


Notes:

When a string is written, “abc”, the String class is instantiated and the new object represents the text “abc”. It isn’t the same as writing: new String(“abc”). When writing just “abc” the String class isn’t necessarily instantiated. If it was already instantiated before creating a String object that represents the same text (simply by writing the text itself e.g. “abc”) then the reference of the already existing object is returned.

Therefore, given the following code,

```
String str1=”abc”;
String str2=”abc”;
if (str1==str2)
        (“equal”);
```

The condition value is true.

And given the following code,

```
String str=new String(“abc”);
String str2=new String(“abc”);
if(str1==str2)
        (”equal”);
```

The condition value is false.

Notes:

The equals() method in class String is an overridden version of the equals() method that String inherits from class Object. The equalsIgnoreCase() method does the comparison ignoring case considerations.

Notes:

All classes extend the class Object (directly or indirectly).

Notes:

Overriding the method toString() is useful not only when debugging. It is also useful when working with a StringBuffer object. Each invocation of the method append() on a StringBuffer object and sending it a reference of other object results in appending the returned text of the toString() method invoked on that object.

Now, the StringBuffer object represents the text: “Haim Michael”.

Notes:

As can be seen, in code 2 lots of String objects are instantiated during the loop. In code 1 the opposite. During the loop in code 2 there isn't any instantiation of new objects.

Sometimes, using the StringBuffer class might improve the performance.

Notes:

As can be seen, in code 2 lots of String objects are instantiated during the loop. In code 1 the opposite. During the loop in code 2 there isn't any instantiation of new objects.

Sometimes, using the StringBuffer class might improve the performance.

Notes:

As can be seen, in code 2 lots of String objects are instantiated during the loop. In code 1 the opposite. During the loop in code 2 there isn't any instantiation of new objects.

Sometimes, using the StringBuffer class might improve the performance.

Notes:

As can be seen, in code 2 lots of String objects are instantiated during the loop. In code 1 the opposite. During the loop in code 2 there isn't any instantiation of new objects.

Sometimes, using the StringBuffer class might improve the performance.

Notes:

As can be seen, in code 2 lots of String objects are instantiated during the loop. In code 1 the opposite. During the loop in code 2 there isn't any instantiation of new objects.

Sometimes, using the StringBuffer class might improve the performance.

Notes:

As can be seen, in code 2 lots of String objects are instantiated during the loop. In code 1 the opposite. During the loop in code 2 there isn't any instantiation of new objects.

Sometimes, using the StringBuffer class might improve the performance.

Notes:

As can be seen, in code 2 lots of String objects are instantiated during the loop. In code 1 the opposite. During the loop in code 2 there isn't any instantiation of new objects.

Sometimes, using the StringBuffer class might improve the performance.

Notes:

An instance of StringTokenizer behaves in one of two ways, depending on whether it was created with the returnDelims flag having the value true or false:

If the flag is false, delimiter characters serve to separate tokens. A token is a maximal sequence of consecutive characters that are not delimiters. If the flag is true, delimiter characters are themselves considered to be part of the tokens.

Notes:

In order to run the following example type the following line in the command line:
java ArgumentsToMain 1 2 3 4