ARRAYS

- A Java array is an Object that holds an ordered collection of elements.

- Components of an array can be primitive types or may reference objects, including other arrays.

- Arrays can be declared, allocated, and / or initialized all in one compact statement.

```
int[] ai;

int[] ia = new int[3];

char ac[] = {'n', 'o', 't'};
```

The length of an array is fixed at creation and cannot be changed, but a new, longer array instance can be assigned to the Object.

The length field of an array object makes the length of the array available.

Java supports arrays of arrays, rather than 2D or multi-dimensional arrays.

```
Example:
public class myArray {
public static void main (String[] args) {

double[] [] mat = {{1., 2., 3., 4.}, {5., 6., 7., 8.},
                   {9., 10., 11., 12.}, {13., 14., 15., 16.}};

for (int y = 0; y < mat.length; y++) {
  for (int x = 0; x < mat[y].length; x++)
    System.out.print(mat[y][x] + " ");
  System.out.println();
}}}
```

What does it print?

Is the array stored in row-major or column-major order?

STRINGS

- The String class provides read-only strings and supports operations on them
- A String can be created implicitly either by using a quoted string (e.g. "grass") or by the concatenation of two String objects, using the + operator.
- A String can also be created using new and a String constructor.

```
String aString = new String();

String bString = new String("grass");
```


String Methods

- length() returns the number of characters
- charAt(i) returns the character at position i.

```
for (int i = 0; i < str.length(); i++)
  System.out.println(charAt(i));
```

- indexOf(char ch) returns the first position of character ch in a String.
- lastIndexOf(char ch) returns the last position of character ch in a String.

String Comparisons

- The method equals returns true if it is passed a reference to a String object with the same contents as a given String.

- The method equalsIgnoreCase works like equals, but ignores upper/lower case distinctions.

- The method compareTo returns an int that is less than, equal to, or greater than zero for comparisons and sorting.

Example: Part of a Binary Search

```
while (lo <= hi) {
  int mid = lo + (hi - lo) / 2;
  int cmp = key.compareTo(table[mid]);
  if (cmp == 0)   return mid;
  else if (cmp < 0)   hi = mid - 1;
  else lo = mid + 1;
}
```


Since you cannot modify existing strings, there are methods to create new strings from existing ones.

- public String substring(int beginIndex, int endIndex)
- public String replace(char oldChar, char newChar)
- public String concat(String str)
- public String toLowerCase()
- public String toUpperCase()
- public String trim()

EXCEPTIONS

- An exception is a condition that arises during execution.
- An exception is thrown when an error condition is encountered.
- It is then caught by code that receives it and deals with it.
- Java exceptions are objects that extend the class Throwable or one of its subclasses, usually the subclass Exception.
- Class Throwable and its subclasses have two constructors, one with no arguments and one with a String argument for an error message.

Defining Exceptions

Programmers can define their own exception types and write code to throw them and catch them.

Example with Attributes and Values:

```
public class NoSuchAttributeException extends Exception
{
  public String attrName;
  public Object newValue;

  NoSuchAttributeException(String name, Object value) {
    super("No attribute named  \"" + name + "\" found");
    attrName = name;
    newValue = value;
  }
}
```

Throwing Exceptions

Methods that will check for errors and throw exceptions use

- the throws clause to tell the compiler what kind of exceptions they may throw
- the throw statement to perform the throwing

Example:

```
public void replaceValue(String name, Object newValue)
  throws NoSuchAttributeException
{
  Attr attr = find(name);
  if (attr == null)
    throw new NoSuchAttributeException(name newValue);
  attr.valueOf(newValue);
}
```


If your method invokes a method that lists a checked exception it its throwable clause, it can do one of three things

- Catch the exception and handle it.
- Catch the exception and map it into one of your own exceptions by throwing an exception of a type declared in your own throws clause
- Declare the exception in your throws clause and let it pass through your method, possibly adding a finally clause that cleans up first.

Catching Exceptions

Syntax:

```
try   <block>
catch (<exception_type>   <identifier>)
      <block>
catch(<exception_type>  <identifier>)
     <block>
  . . .
finally  <block>
```

Example:

```
try{
   attributedObj.replaceValue("Age", new Integer(8));
    }
catch(NoSuchAttributeException e) {

   Attr attr = new Attr(e.attrName,  e.newValue);
   attributedObj.add(attr);
}
```


Finally

- The finally clause of a try statement provides a mechanism for executing a section of code whether or not an exception is thrown.

- This example closes a file when its work is done, even if an error occurs:

```
public boolean searchFor(String file, String word)
    throws StreamException
{
   Stream input = null;
   try {
        input = new Stream(file);*
        while (!input.eof())
          if (input.next() == word)  return* true;
        return* false; }
   *finally {
             if (input != null) input.close();
             }
}
```

* This can fail. * Finally is executed before any return.