Lab #1
Introduction to Java
CMP(N) 306

Cairo University
Faculty of Engineering
Computer Engineering Department

Agenda

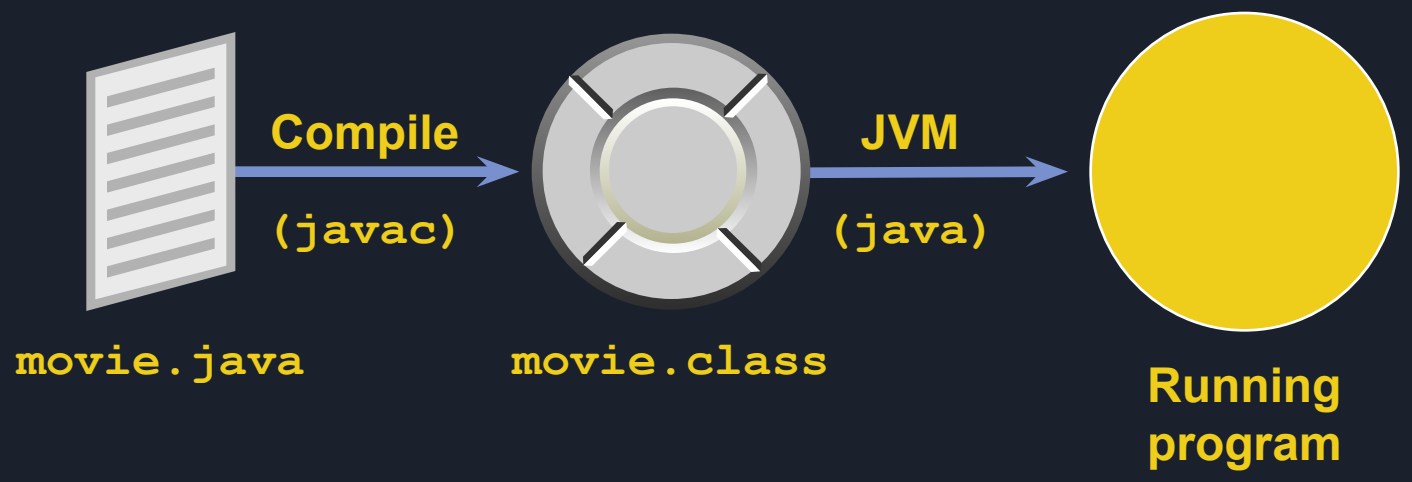
- Course Information
- Java
  - What is Java?
  - How do Java based programs run?
  - C++ vs Java
  - Syntax
  - Strings & Arrays

What Is Java?

- Object-oriented programming language
- Platform independent
- Distributed
- Multithreaded
- Robust and secure

Platform Independence

- Java code is stored as a `.java` file.
- The .java program is compiled into `.class` files.
- The `.class` file contains *Java bytecodes*, which are platform-independent machine instructions.
- Bytecodes are interpreted at run time by the Java Virtual Machine (JVM) at run time to the native instruction set based on the current platform.



How the JVM Works

- The JVM class loader loads all required classes.
- The JVM verifier checks for illegal bytecodes.
- The JVM memory manager releases memory back to the OS. The process the JVM uses to manage dereferenced objects is called *garbage collection*.

JRE and JDK

Java Runtime Environment (JRE)

- Java Runtime Environment is the implementation of ***JVM***. It contains ***JVM***, class libraries, and other supporting files. If you want to run any java program, you need to have ***JRE*** installed in the system

Java Development Kit (JDK)

- Java Developer Kit contains tools needed to develop the Java programs, and ***JRE*** to run the programs. You need ***JDK***, if you want to write your own programs, and to compile them. For running java programs, ***JRE*** is sufficient.

Compiling and Running a Java Application

- To compile a `.java` file:

```
prompt> javac SayHello.java
... compiler output ...
```

- To execute a `.class` file:

```
prompt> java SayHello
Hello world
prompt>
```

C++ VS Java

	C++	Java
Compilation Output	C++ compiler converts source code into machine level language	Java interpreter converts the source code into byte code
Platform Dependence	platform dependent	platform independent
Garbage Collection	Responsibility of the programmer	Auto built-in garbage collection
Pointers	supported	not supported. But in later versions, the promoters began providing "Restricted pointers”
Object Oriented	Allows OOP	Everything (except primitive types) is an object in Java (everything gets derived from java.lang.Object).

Syntax

Naming Conventions

Naming conventions include:

- Filenames

  `Customer.java, RentalItem.java`

- Class names

  `Customer, RentalItem, InventoryItem`

- Method names

  `getCustomerName(), setRentalItemPrice()`

Naming Conventions

- Standard for variables

  `customerName, customerCreditLimit`

- Standard for constants

  `MIN_WIDTH, MAX_NUMBER_OF_ITEMS`

- Use uppercase and lowercase characters
- Numerics and special characters

Class Definition

```
public class Customer {
 // Instance variables
 String customerName;
 String customerPostalCode;
 float customerAmountDue;
 ...
 // Instance methods
 float getAmountDue (String cust) {
  ...
 }
 ...
}
```

Declaration

Instance variable

Instance method

Method Definition

```
float getAmountDue (String cust) {
 // method variables
 int numberOfDays;
 float due;
 float lateCharge = 1.50;
 String customerName;
// method body
 numberOfDays = this.getOverDueDays();
 due = numberOfDays * lateCharge;
 customerName = getCustomerName(cust);
 return due;
}
```

Declaration

Method variables

Method statements

Return

Variable Names

- Variable names must start with a letter of the alphabet, an underscore, or a $.
- Other characters may include digits.

```
a   item_Cost
itemCost ✓ _itemCost
item$Cost itemCost2
```

```
item#Cost     item-Cost
item*Cost ✗   abstract
2itemCost
```

- Use meaningful names for variables; for example, *`customerFirstName, ageNextBirthday`*.

Reserved Keywords

```
boolean
byte
char
double
float
int
long
short
void
```

```
false
null
true
```

```
abstract
final
native
private
protected
public
static
synchronized
transient
volatile
```

```
break
case
catch
continue
default
do
else
finally
for
if
return
switch
throw
try
while
```

```
class
extends
implements
interface
throws
```

```
import
package
```

```
instanceof
new
super
this
```

Primitive Data Types

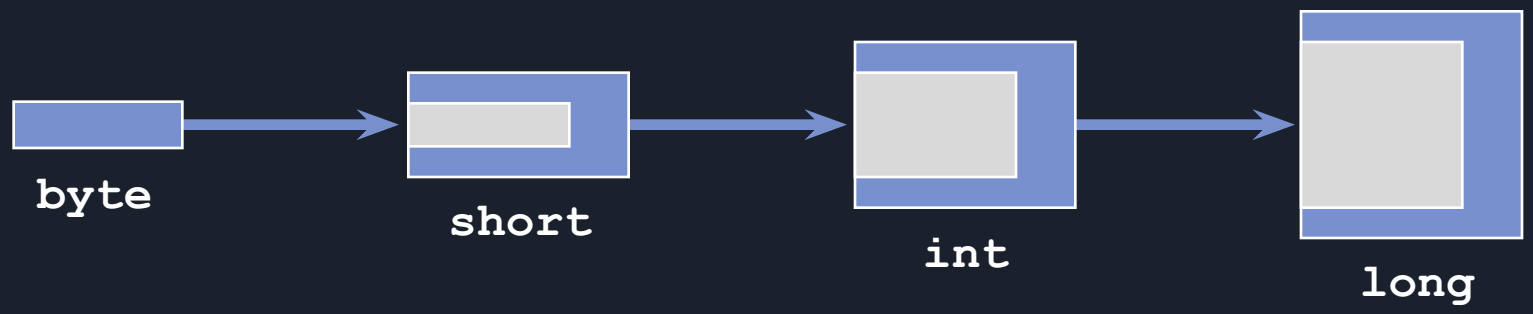
Integer	Floating Point	Character	True False
byte short int long	float double	char	boolean
1,2,3,42 07 0xff	3.0 .3337 4.022E23	'a' '\141' '\u0061' '\n'	true false

Non-Primitive Data Types: Primitive Wrapper Classes

Primitive	Wrapper Class	Constructor Argument
boolean	Boolean	boolean or String
byte	Byte	byte or String
char	Character	char
int	Integer	int or String
float	Float	float, double or String
double	Double	double or String
long	Long	long or String
short	Short	short or String

Conversions and Casts

- Java automatically converts a value of one numeric type to a larger type.



- Java does not automatically “downcast.”

Increment and Decrement

- The ++ operator increments by 1:

```
int var1 = 3;
var1++;            // var1 now equals 4
```

- The ++ operator can be used in two ways:

```
int var1 = 3, var2 = 0;
var2 = ++var1;    // Prefix:  Increment var1 first,
                  //          then assign to var2.
var2 = var1++;    // Postfix: Assign to var2 first,
                  //
```

- The -- operator decrements by 1.

Logical Operators

Results of Boolean expressions can be combined by using logical operators:

`&&` `&`	and (with / without short-circuit evaluation)
`||` `|`	or (with / without short-circuit evaluation)
`^`	exclusive or
`!`	

```
int var0 = 0, var1 = 1, var2 = 2;
boolean res = true;
res = (var2 > var1) & (var0 == 3);   // now false
res = !res;                          // now true
```

String Concatenation

The + operator creates and concatenates strings:

```
String name = "Paul ";
String lastName = "Edward";
String fullName;
name = name + lastName;    // name is now
                           //"Paul Edward"
fullName = name;
```

More on the `for` Loop

- **Initialization and iteration can consist of a list of comma-separated expressions:**

```
for (int i = 0, j = 10; i < j; i++, j--) {
    System.out.println("i = " + i);
    System.out.println("j = " + j);
}
```

Transfer Control Flow Statements

break statement: breaks out of a loop or switch statement and transfers control to the first statement after loop body or `switch` statement

```
...
while (age <= 65) {
    balance = (balance+payment) * (1 + interest));
    if (balance >= 250000)
        break;
    age++;
}
...
```

Transfer Control Flow Statements

continue statement : Can be used only in loops. It abandons the current loop iteration, and jumps to the next loop iteration

```
...
for (int year = 2000; year < 2099; year++) {
    if ((year % 100 == 0) && (year % 400 != 0))
        continue;
    if (year % 4 == 0)
        System.out.println(year);
}
...
```

Transfer Control Flow Statements

Labeled break and continue Statements: Can be used to break out of nested loops, or continue a loop outside the current loop

```
outer_loop:
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 5; j++) {
        System.out.println(i, j);
        if (i + j > 7)
            break outer_loop;
    }
}
...
```

What Is a String?

- A string is a sequence of characters.
- The `String` class represents all strings in Java.
- `String` objects are read-only; their values cannot be changed after creation.
  - The following statements point the object reference `str` to a new location in memory, rather than changing the contents of the string:

```
String str = "Action";
  str = "Comedy";
```

Useful Methods in String Class

Method	Usage
`int length();`	find the length of a string
`char charAt(int index);`	find the character at a specific index
`String substring (int beginIndex, int endIndex);`	return a substring of a string
`String toUpperCase();` `String toLowerCase();`	convert to uppercase or lowercase
`;()String trim`	trim whitespace

`;(int indexOf (String str`
`int lastIndexOf`
`;((String str`
	find the index of a substring

How to Compare Two Strings

- Use `equals()` if you want case to count:

```
String passwd = connection.getPassword();
if (passwd.equals("fgHPUw"))... // Case is important
```

- Use `equalsIgnoreCase()` if you want to ignore case:

```
String cat = getCategory();
if (cat.equalsIgnoreCase("Drama"))...
          // We just want the word to match
```

- Do not use `==`.

How to Produce Strings from Other Objects

- Use `Object.toString()`.
- Your class can override `toString()`:

```
public Class Movie {...
    public String toString {
        return name + " (" + Year + ")";
    }...
```

- `System.out.println()` automatically calls an object's `toString()` method:

```
Movie mov = new Movie(...);
System.out.println("Title Rented: " + mov);
```

How to Produce Strings from Primitives

- Use `String.valueOf()`:

```
String seven = String.valueOf(7);
String onePoint0 = String.valueOf(1.0f);
```

- There is a version of `System.out.println()` for each primitive type:

```
int count;
...
System.out.println(count);
```

How to Produce Primitives from Strings

- Use the primitive wrapper classes.
- There is one wrapper class for each primitive type; for example:
  - `Integer` wraps the `int` type.
  - `Float` wraps the `float` type.
- Wrapper classes provide methods to convert a String to a primitive.

```
String qtyVal = "45";
String priceVal = "340.5F";
int qty = Integer.parseInt(qtyVal);
float price = Float.parseFloat(priceVal);
```

How to Change the Contents of a String

- Use the `StringBuffer` class for modifiable strings of characters:

```
public String reverseIt(String s) {
  StringBuffer sb = new StringBuffer();
  for (int i = s.length() - 1; i >= 0; i--)
    sb.append(s.charAt(i));
  return sb.toString();
}
```

- Use `StringBuffer` if you need to keep adding characters to a string.

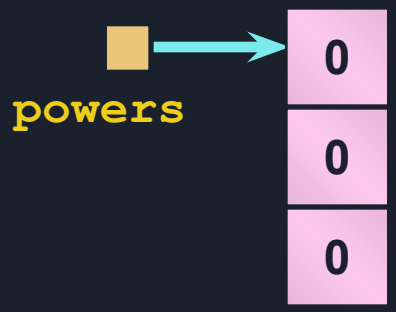
How to Create an Array of Primitives

1. Declare the array.

```
int[] powers;
    Or
int powers[];
```

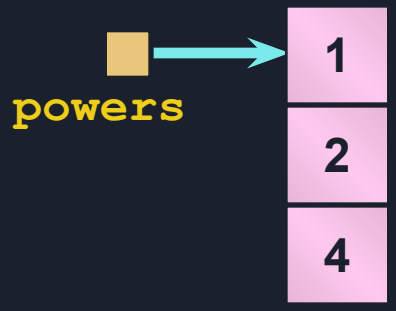
2. Create the array object.

```
powers = new int[3];
```



3. Initialize the array elements (optional).

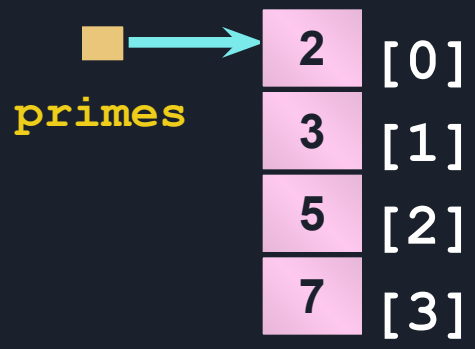
```
powers[0] = 1;
```



Initializing the Array Elements

```
type[] arrayName = {valueList};
```

```
int[] primes = {2, 3, 5, 7};
```

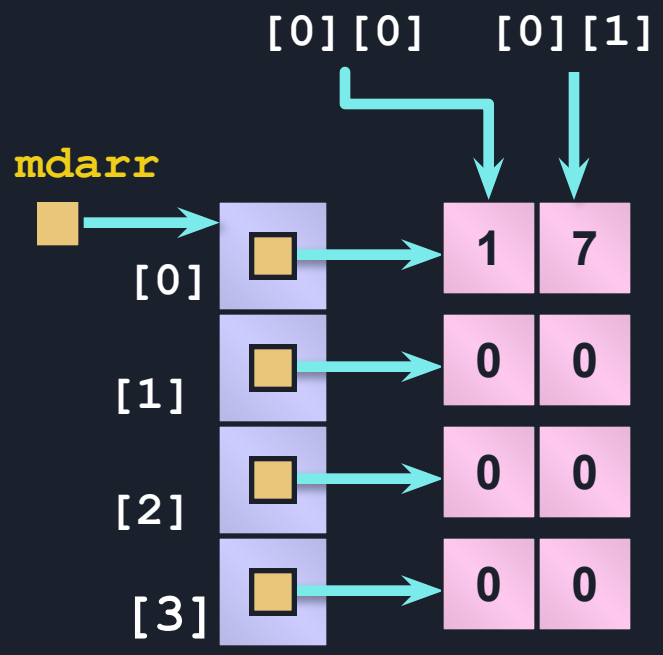


Multidimensional Arrays

Java supports arrays of arrays:

```
type[][] arrayname = new type[n1][n2];
```

```
int[][] mdarr = new int[4][2];
mdarr[0][0] = 1;
mdarr[0][1] = 7;
```



How to Implement Resizable Arrays

The `Vector` class implements a resizable array of any type of object:

- Creating an empty vector:

```
Vector members = new Vector();
```

- Creating a vector with an initial size:

```
// Create a vector with 10 elements. The vector
// can be expanded later.
Vector members = new Vector(10);
```

How to Modify a Vector

- Add an element to the end of the vector:

```
String name = Movie.getNextName();
vector.addElement(name);
```

- Add an element at a specific position:

```
// Insert a string at the beginning
vector.insertElementAt(name, 0);
```

- Remove the element at a specific index:

```
// Remove the first element
vector.removeElementAt(0);
```

How to Access a Vector

- Get the first element:

```
String s = (String)vector.firstElement();
```

- Get an element at a specific position:

```
String s = (String)vector.elementAt(2);
```

- Find an object in a vector:

```
int position = vector.indexOf(name);
```

- Get the size of a vector:

```
int size = vector.size();
```

The Basics: Standard Output

Understanding `System.out.println()`

- There is no package called `System` with a class named `out` and a `println()` method.
- `System` is a class.
- `System` has a static instance variable called `out`.
- `out` represents another class.
- `out` is a `PrintStream` object.