Introduction to Java

Getting Started

What is Java?

- A platform independent (JVM), highly portable programming language
- Object-oriented (supports objects, classes and inheritance)
- Interpreted language (Java source code -> bytecode -> JVM)
- Strongly-typed language (checked at compile and run time)
- Automatic memory management (allocation, de-allocation)

JVM - Java Virtual Machine

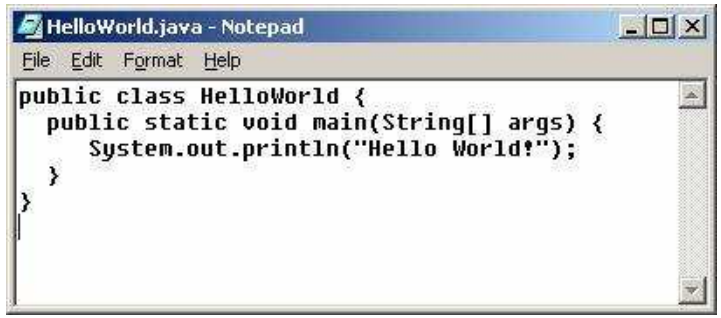
- An implementation of the JVM Specification, interprets compiled Java binary code (bytecode)
- Programs are compiled by the Java compiler into bytecode
- JVM interprets compiled Java bytecode and executes the Java program

JDK & JRE

- Java Development Kit and Java Runtime Environment
- JRE consists of JVM and Java class libraries
- JDK consists of JRE and development tools (compiler, debugger, disassembler, etc.)

Source Code

- Unit of compilation in the source file (containing class and interface definitions)
- Placed in text files with .java extension
- Typically one class or interface per source file



Packages

- Grouping of classes and related sub-packages into one based on functionality
- Unique namespaces
- Hierarchical system

- Package name structure should match corresponding directory structure
- The package keywords describes which package the class will belong to
- Packages have simple and specialized fully qualified names.
- By convention, they use reversed internet domain names (com.epam.jdi)
- Access restriction
    - Default access is also known as package access, any other class within the same package will have full access
    - Protected access: Members can only be accessed within their own packages
    - Sub-packages are considered totally different from their parent packages access-wise

Using Package Members

- Types that comprise a package are known as the package members
- How to use a public package member from outside its package:
    - Importing the package member
    - Import the member's entire package
    - Referring to the member by its fully qualified name (com.epam.jdi.Tester xyz = new com.epam.jdi.Tester();)
- Importing works with the import keyword (import com.epam.jdi.Tester; Tester yxz = new Tester();)
- Importing a member's entire package:
    - works with the asterisk (import com.epam.jdi.*;)
    - Doesn't import members from sub-packages
    - At most one asterisk can be used

Class-path and Import

- Most Java apps are built from many classes, Java needs to locate those class files:
    - Compiler needs class files at compile time
    - JVM needs class files at run-time
- Class-path:
    - List of directories or JAR files containing class files
    - Checked to find required classes
    - May be specified OS-level by setting the CLASSPATH (set CLASSPATH=C:\yxz\classes;C:\abc\lib\utils.jar)

Variables

- Variables are entities in Java that can hold a value
- Java defines the following kinds: Instance variables, Class variables, Local variables, Parameters
- All variables must be declared before first use
- Local variables go out of scope at the end of the enclosing block
- Multiple variables can be specified in one declaration
- Declaration can be entended with first value assigment

```
int answer = 42, lucky = 7;
double betterAnswer = 42.0;
String name = "Peter";
```

Primitive Data Types

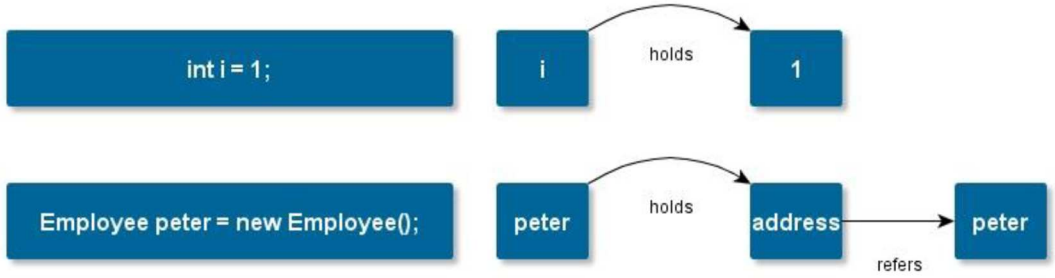
- A primitive is a simple non-object data type that represents a single value
- Primitives:
  - The boolean type
  - Integral types: byte, short, int long
  - Floating-point types: float, double
  - Textual type: char

Literals

- Because primitives are non-objects to initialize them literals must be used
- A literal is the source code representation of a fixed value
- Literals are represented directly in the code without requiring computation
- Integer literals: int i = 10000; long l = 10000L;
- Floating-point literals: double d = 123.4; float f = 123.4f;
- Character and String literals: char c = ’C’; String s = „string”;
- Null literal: Value for any reference type and can be assigned to any variable. Used to represent uninitialized state.
- Class literal: Refers to the object (of type Class) that represents the type itself;

Reference Data Types

- Reference values are pointers to objects
- Primitive variables store primitive values



Arrays

- A special object. Ordered collection of primitives or references
- An array variable is a reference that points to an array object
- Declaring the array variable creates the reference variable, the array object must be instantiated seperately
- Once instantiated the array object contains a block of memory location for the individual elements
- If the elements are not explicitly initialized, they will be set to zero
- Arrays can be created with a size that is determined dynamically, but once created the size is fixed
- To create: declaration -> construction -> initialization

```
int[] ints;          // Declaration to the compiler
ints = new int[25];  // Runtime construction
```

```
String[] names;

names = new String[] {"Joe", "Jane", "Herkimer"};
```

Working with Arrays

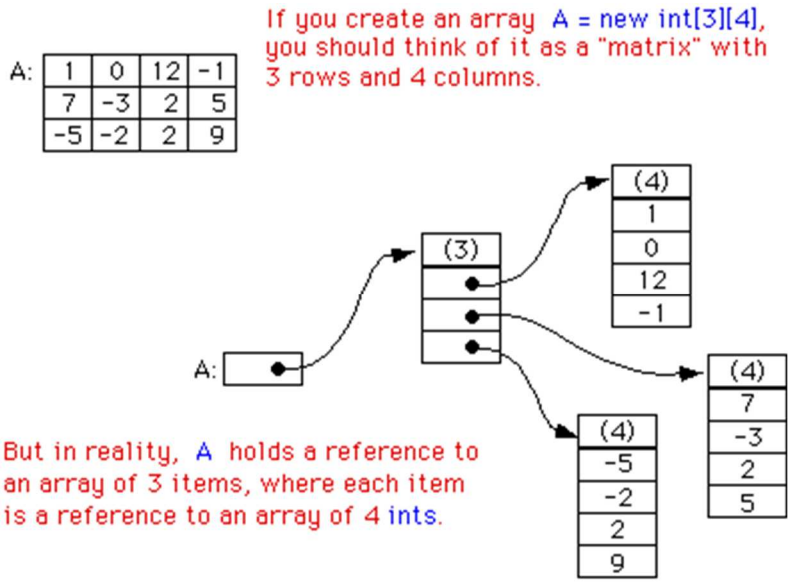
- As objects arrays have properties
- Length: Read-only, contains the capacity of the array (int x = names.length;)
- Index: Array items are access through the array index. Items are numbered from 0.

```
String[] names = new String[3];

names[0] = "Peter";
names[1] = "Dave";
names[2] = "Mike";
```

Multi-Dimensional Arrays

- A two-dimensional array is a single array of array reference variables
- Each of which poits to a single dimensional array



Keywords and Reserved Words

- keywords are predefined identifiers reserved by Java for a specific purpose
- goto and const are reserved words (in Java they don't mean what one would expect)

Identifiers

- They are the names of:
  - variables
  - methods
  - classes
  - packages
  - interfaces
- Must be composed of:
  - letters
  - numbers
  - underscore sign
  - dollar sign
- They are case sensitive

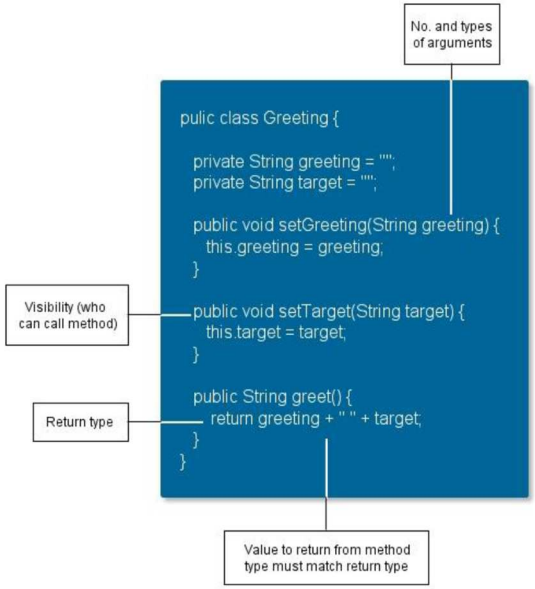
Java Naming Convention

- Classes and interfaces
    - First letter should be capitalized
    - Then CamelCase if several words are linked together
    - Classes are typically nouns: Account, Dog
    - Interfaces are typically adjectives: Runnable, Serializable

- Methods:
    - First letter should be lowercase
    - camelCase rule afterwards
    - Typically verb-noun pairs: getPrice, setCustomerName, doCalculation
- Variables:
    - First letter should be lowercase, then camelCase rule
    - Examples: buttonWidth, accountBalance, myString
- Constants:
    - Should be markes as static and final
    - Uppercase letters with underscore separator
    - Examples: MIN_HEIGHT, TEXT_WIDTH

Methods

- A callable reusable unit of code
- To define a method the user needs to provide: Name, return type, arguments with types
- A method can return a value of any type or nothing (void)



Overloading

- Method names could be overloaded:
  - Normally within same class
  - Formal parameter list must be different
  - Return type may be different

```
private int max(int n1, int n2) {
  return (n1 > n2) ? n1 : n2;
}
private int max(int n1, int n2, int n3) {
  int n = (n1 > n2 ) ? n1 : n2;
  return (n > n3) ? n : n3;
}
...
int maxOf2 = max(100, 33);
int maxOf3 = max(20, 30, 40);
...
```

Varargs

- Three dots after the last parameter's type indicate that the last argument may be passed as an array or as a sequence of arguments
- Example: public static String format(String pattern, Object... arguments);

Static Methods

- Methods with static modifier in their declarations
- Should be invoked with the class name, without the need for creating an instance of the class
- Static methods can acces static variables and static methods directly
- They cannot access instance variables or instance methods direclty, they must use an object reference
- Static methods cannot use the this keyword, as there is no instance for this to refer to

Example:

```
class Foo {
    int i;

    public Foo(int i) {
       this.i = i;
    }

    public static String method1() {
       return "An example string that doesn't depend on i (an instance variable)";
    }

    public int method2() {
       return this.i + 1; // Depends on i
    }
}
```

The main Method

- The JVM begins execution of a class by invoking this method
- Usage: public static void main(String[] args)

Command Line Arguments

- The only parameter of the main method, an array of strings
- The JVM copies runtime arguments from the program's command line
- Accepts any number of arguments from the command line

Assess Modifiers

- public
- private
- protected
- default

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Java Operators

- Casting
  - Java allows to make explicit casts from one primitive type to another, with the exception of the boolean
  - Narrowing conversion: int i = (int)12.5f;
  - Widening conversion: float f = i;
- instanceof
  - Type comparison operator, checks for assigment compatibility
  - null is not an instance of anything!
- new
  - The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory
  - The new operator is followed by a call to a constructor, which initializes the new object
  - The new operator requires a single, postfix argument
  - Example: Point ofOrigin = new Point(42,42);

```
public static void main(String args[]) {
  A a = new A();

  if(a instanceof A) {
    System.out.println("a is instance of A");
  }
}
```

Arithmetic Operators

- Unary plus and minus: +1 -1
- Product, division and modulo: A*B A/B A%B
- Addition and substraction: A+B A-B

Logical and Operators

- Logical expressions yield boolean values
  - Logical equality or ineqality: A == B A != B
  - Logical AND, OR: A && B A || B
  - Logical NOT: !A

Miscellaneous Operators

- Conditional operator: condition ? true : false;
- Increment and decrement operators (available in pre- or postfix form): ++ --

Expressions

- A construct made up of variables, operators or method invocations
- Constructed according tot the synthax of the language: anArray[0] = 100 index = 1

Statements

- A statement forms a complete unit of execution
- Following types of expression can be made into a statement by terminating the expression with a semicolon:
    - Assigment expressions
    - Any use of ++ or --
    - Method invocations
    - Object creation expressions
- Examples: int index = 1; System.out.println(„Index of element: „ + index);

Control Flow Statements - if

- The else clause is optional and may be also followed by another if statement
- The statements can be compound block statements
- The expression must yield a boolen value

```
if ( n > 0 ) {
  System.out.println(" N is greater than zero ");
}
```

Control Flow Statements - switch

- The expression must yield ordinal or enumerated type
- From Java 7 String-based switch is possible
- Body of statements: switch block
- One or more case or default labels
- The break statement: terminates the enclosing switch statement. Lack of it causes fall-through
- The default section: handles all cases that are not explicitly handled

```
if ( x==0 && y==0 ) {
  System.out.println("At the origin");
} else if ( x>0 && y>0 ) {
  System.out.println("Top right quadrant");
} else if ( x<0 && y<0 ) {
  System.out.println("Bottom left quadrant");
} else {
  System.out.println("Inside Q1 and Q2");
}
```

Control Flow Statements - looping

- Main forms of loop control flow:
  - for loop
  - while loop
  - do/while loop
- The keyword break is used to terminate a loop (in the case of embedded loops, it always terminates the innermost one)
- The keyword continue is used to force a loop into the next iteration

```
int day;
String dayName;
...
switch (day) {
  case 1:
      dayName = "Monday";
      break;
  case 2:
      dayName = "Tuesday";
      break;

   ...

  default:
      dayName = "Sunday";
      break;
}
```

Control Flow Statements - for loop

- The for statement provies a compact way to iterate over a range of values
- The initialization expression initializes the loop; it's executed once,
- As the loop begins when termination expression evaluates to false, the loop terminates
- increment expression is invoked after each iteration
- for(initialization; condition; update) statement;

Control Flow Statements - for-each loop

- This form is designed for iteration through each element of:
  - Collections
  - Arrays
  - Instances of classes implementing the java.lang.Iterable<T> interface

```
public static void main(String[] args) {
    int[] numbers =
      {1,2,3,4,5,6,7,8,9,10};
    for (int item : numbers) {
      System.out.println("Count is: " + item);
    }
}
```

Control Flow Statements - while loop

- The while statement evaluates a condition expression
  - Must return a boolean value
- The while statement continues testing the expression and executing its block until the expression evaluates to false
- One can implement an infinite loop using the while statement (for and do-while can also be used for this purpose)
- Example while (x < 100) { //dosomething; x++;}

Control Flow Statements - do-while loop

- The do-while evaluates its expression after the execution of each iteration of the loop
- Statements within the do block are always executed at least once

Comments

- Comments are not part of the program itself - does not affect the programming logic
  - Used only for documentation
- Comments out the rest of the line – "line comment" - //comment
- Comments out several lines (no nesting) – "block comment" - /* line1 line2 line3 */
- Comment to semi-automate code documentation – "javadoc comment"
- Recognised by javadoc standard utility (and by many IDEs in real-time)

```
/**
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
*
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
```