Introduction to JAVA
JAVA Basics
Introduction

- Developed by James Gosling at Sun Microsystems.
- It was introduced in 1995.
- Is one the fastest growing programming technologies of all time.
The java environment

- **code**
  - Java code `.java`

- **compile**
  - Java compiler `javac`

- **result**
  - Java bytecode `.class`

**Runs on**
- Windows Running JVM
- MAC Running JVM
- Linux Running JVM

Slides prepared by: Farzana Rahman
The Java programming environment

- Java programming language specification
  - Syntax of Java programs
  - Defines different constructs and their semantics

- *Java byte code*: Intermediate representation for Java programs

- *Java compiler*: Transform Java programs into Java byte code

- *Java interpreter*: Read programs written in Java byte code and execute them

- *Java virtual machine*: Runtime system that provides various services to running programs

- *Java programming environment*: Set of libraries that provide services such as GUI, data structures, etc.

- *Java enabled browsers*: Browsers that include a JVM + ability to load programs from remote hosts
Bytecode

• Java programs are translated into an intermediate language called **bytecode**.

• Bytecode is the same no matter which computer platform it is run on.

• Bytecode is translated into native code that the computer can execute on a program called **Java Virtual Machine (JVM)**.

• The Bytecode can be executed on any computer that has the JVM. Hence Java’s slogan, “Write once, run anywhere”.
Important things to know

• Java is CASE SENSITIVE!!
• Whitespace is ignored by compiler
• File name has to be the same as class name in file
• Need to import necessary class definition
How to install JAVA

• The Java Development Kit (JDK) is a collection of software available at no charge from Sun Microsystems, Inc. The v1.3 download is available at java.sun.com.
How to get it running

• Text in **hello.java** file
  
  – The class that holds the main method, should be named **hello**

• To compile:
  
  – javac hello.java

• To run:
  
  – java hello
Sample Java program

class Hello {
    public static void main ( String[] args ) {
        System.out.println("Hello World!");
    }
}
Data types

• A data type is a scheme for representing values.
• An example is int which is the integer, a data type.
• Values are not just numbers, but any kind of data that a computer can process.
• The data type defines the kind of data that is represented by a variable.
• As with the keyword class, Java data types are case sensitive.
Primitive Java Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size (byte)</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
<td>true or false</td>
</tr>
<tr>
<td>char</td>
<td>2 (Unicode)</td>
<td>A-Z, a-z, 0-9, etc.</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>(about)–2 million to 2 million</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
<td>(about)-10E18 to 10E18</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>-3.4E38 to 3.4E18</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>-1.7E308 to 1.7E308</td>
</tr>
</tbody>
</table>

- There are only eight primitive data types.
- A programmer cannot create new primitive data types.
Objects

• All data in Java falls into one of two categories: primitive data and objects.

• There are only eight primitive data types.

• Any data type you create will be an object. An object is a structured block of data. An object may use many bytes of memory.

• The data type of an object is its class.

• Many classes are already defined in the Java Development Kit.

• A programmer can create new classes to meet the particular needs of a program.
How to define expressions?

• Arithmetic: +, -, *, /, %, =

\[ 8 + 3 \times 2 / 4 \]

Use standard precedence and associativity rules

• Predicates: ==, !=, >, <, >=, <=

```java
public class Demo {
    public static void main (String[] argv) {
        boolean b;
        b = (2 + 2 == 4);
        System.out.println(b);
    }
}
```

Slides prepared by: Farzana Rahman
Variables

• A declaration of a variable is where the program allocates memory for the variable.

• The declaration in the example program requested a 32 bit section of memory which will use primitive data type int and will be named states.

• A variable cannot be used in a program unless it has been declared.

• A variable can be declared only once in a particular section of code.
Variables (Cont.)

• Variables:
  – Name
  – Type
  – Value

• Naming:
  – May contain numbers, underscore, dollar sign, or letters
  – Can not start with number
  – Can be any length
  – Reserved keywords
  – Case sensitive
**Initialization**

- If no value is assigned prior to use, then the compiler will give an error.
- Java sets primitive variables to zero or false in the case of a boolean variable.
- All object references are initially set to null.
- An array of anything is an object:
  - Set to null on declaration.
  - Elements to zero false or null on creation.
Declarations

```java
int index = 1.2;        // compiler error
boolean retOk = 1;      // compiler error
double fiveFourths = 5 / 4;  // no error!
float ratio = 5.8f;     // correct
double fiveFourths = 5.0 / 4.0;  // correct
```

- 1.2f is a float value accurate to 7 decimal places.
- 1.2 is a double value accurate to 15 decimal places.
Assignment statements

• variables are expected to vary by having new values placed into them as the program runs. An assignment statement is one way to change the value of a variable.

```
public class example {
    public static void main ( String[] args ) {
        int states; // a declaration of a variable
        states = 50; // an assignment statement
        System.out.println(“The variable states contains: “ + states);
    }
}
```

The assignment statement puts the value 50 into the variable. It means, when the program is executing there will be a 32 bit section of memory that gets value 50.
Assignment statement syntax

• Assignment statements look like this:

    \( \text{variableName} = \text{expression}; \)

• The equal sign "=" means "assignment."

• \( \text{variableName} \) is the name of a variable that has been declared somewhere in the program.

• \( \text{expression} \) is an expression that has a value.

• An assignment statement asks for the computer to perform two steps, in order:
  
  • 1. Evaluate the expression (that is: calculate a value.)
  
  • 2. Store the value in the variable
Math

• Unary
  
  int x = -9;

• Regular math (+,-,*,/)
  
  int y = 3+x;

• % modulo operator
Incrementing

• Increment and Decrement

• $i++$ equivalent to $i = i + 1$;

• Can also do $++i$, which uses $i$ before incrementing it.

• Decrementing: $i--$;
Type casting

int n = 40;
Wrong : byte b = n;
Right:   byte b = (byte) n;

Type casting converts to target type
Type casting (Cont.)

• Type char is stored as a number. The ASCII value of the character.

• A declaration of :
  – char c = ‘B’;
    stores the value 66 in location c
    can use its value by casting to int
Assignment

- `+=`
- `-=`
- `*=`
- `/=`
- `%=`
Boolean expressions

- boolean b
  b will be either true (1) or false (0)

- Logical operations: !(not), && (and) || (or)

- boolean a,b;
  a = true;
  b = false;
  System.out.println("a && b is " + (a && b));
Arithmetic operators

• An arithmetic operator is a symbol that performs some arithmetic.

• If several operators are used in an expression, there is a specific order in which the operations are applied.

• Operators of higher precedence will operate first.
Important !! regarding precedence

Evaluate Equal Precedence from Left to Right

• When there are two (or more) operators of equal precedence, the expression is evaluated from left to right.

Parentheses

• Expressions within matched parentheses have the highest precedence.

Nested Parentheses

• Sometimes, in a complicated expression, one set of parentheses is not enough. In that case use several nested sets to show what you want. The rule is:

The innermost set of parentheses is evaluated first.
Arithmetic operators (Cont.)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Unary minus</td>
<td>Highest</td>
</tr>
<tr>
<td>+</td>
<td>Unary plus</td>
<td>Highest</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Middle</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>Middle</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>Middle</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>Low</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Low</td>
</tr>
</tbody>
</table>
Relational operators

• == equality
• != inequality
• > greater than
• < less than
• >= greater than or equal to
• <= less than or equal to
Strings

• Not a primitive class, its actually something called a wrapper class
• To find a built in class’s method use API documentation.
• String is a group of char’s
• A character has single quotes
  – char c = ‘h’;
• A String has double quotes
  – String s = “Hello World”;
• Method length
  – int n = s.length;
How to use Strings

class hello
{
   public static void main (String [] args)
   {
      String s = "Hello World\n";
      System.out.println(s); //output simple string
   }
}//end class hello

Slides prepared by : Farzana Rahman
JAVA I/O
Input and Output

• There are no standard statements in Java for doing input or output.

• All input and output is done by using methods found in classes within the JDK.

• Objects in memory communicate in different ways. Each way is a method, and using each method results in a different reaction to the object being used.

• The java.io package is the package used for I/O.

• A package is a collection of classes which may be used by other programs.
IO streams

• In Java, a source of input data is called an input stream and the output data is called an output stream.

• Input data is usually called reading data; and output data is usually called writing data

Commonly used IO Streams

• System.in -- the input stream form the keyboard
• System.out -- the output stream for normal results to the terminal
• System.err -- the output stream for error messages to the terminal
Import a package

• The line import java.io.*; says that the package java.io will be used. The * means that any class inside the package might be used.
Control structures

• Java executes one statement after the other in the order they are written

• Many Java statements are flow control statements:
  Alternation: if, if else, switch
  Looping: for, while, do while
  Escapes: break, continue, return
The if - branching statement

```c
if ( x < y )
{
    x = y;
}
else
{
    x = 88;
}
```
If/Else

if  (logic condition)  
{
   something
}
else if (logic condition)  
{
   something
}
else
{
   something else
}
Nested IF

```java
if ( x < 0 ) {
    System.out.println( "x is negative" );
}
else {
    if ( x > 0 ) {
        System.out.println( "x is positive" );
    }
    //end if x > 0
    else {
        System.out.println( "x is zero" );
    }
} //end else x >=0
```
Switch (variable)
{
    case (1): something;
    break;
    case (23): something;
    break;
    default: something;
}

Slides prepared by: Farzana Rahman
The for loop

• **Loop n times**
  
  ```
  for ( i = 0; i < n; n++ ) {
      // this code body will execute n times
      // from 0 to n-1
  }
  ```

• **Nested for:**
  
  ```
  for ( j = 0; j < 10; j++ ) {
      for ( i = 0; i < 20; i++ ){
          // this code body will execute 200 times
      }
  }
  ```
while loops

```java
while(response == 1) {
    System.out.print( "ID =" + userID[n]);
    n++;
    response = readInt( "Enter ");
}
```

What is the minimum number of times the loop is executed?
What is the maximum number of times?
do {… } while loops

do {
    System.out.print( "ID =" + userID[n] );
    n++;
    response = readInt( "Enter " );
}while (response == 1);

What is the minimum number of times the loop is executed?
What is the maximum number of times?
Break

- A break statement causes an exit from the innermost containing while, do, for or switch statement.

```java
for ( int i = 0; i < maxID, i++ ) {
    if ( userID[i] == targetID ) {
        index = i;
        break;
    }
} // program jumps here after break
```
Continue

- Can only be used with while, do or for.
- The continue statement causes the innermost loop to start the next iteration immediately

```java
for ( int i = 0; i < maxID; i++ ) {
    if ( userID[i] != -1 ) continue;
    System.out.print( "UserID " + i + " :" + userID);
}
```
Exceptions

• Java exception object.

• java.io.Exception
  most general one.
  Some exception like in Throwable class define methods to get the message.
try….catch blocks.

Try
{
    .......
}
catch ( IOException v)
{
    .........
}
System.out.println

- println is a method in the Printstream class.
- Defined:
  - public void println(String x)

  can be any type of string or combination string using addition to join parts.
  Example:
  println(“hello “ + “world “ + x);
System.exit()

- One method in java.lang.System
- Defined:
  
  public static void exit ( int status)

- Terminates currently running Java VM
- Status is status code, non zero will usually mean something abnormal.
- Used at end to indicate success, or in middle to signal problems.
ARRAY
Arrays

- An array is a list of similar things
- An array has a fixed:
  - name
  - type
  - length
- These must be declared when the array is created.
- Arrays sizes cannot be changed during the execution of the code
Arrays example

myArray = \[
\begin{array}{cccccccc}
3 & 6 & 3 & 1 & 6 & 3 & 4 & 1 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

myArray has room for 8 elements the elements are accessed by their index in Java, array indices start at 0
Declaring arrays

```java
int myArray[];
    declares `myArray` to be an array of integers
myArray = new int[8];
    sets up 8 integer-sized spaces in memory, labelled
        `myArray[0]` to `myArray[7]`
int myArray[] = new int[8];
    combines the two statements in one line
```
Assigning values

- refer to the array elements by index to store values in them.
  
  ```
  myArray[0] = 3;
  myArray[1] = 6;
  myArray[2] = 3; ...
  ```

- can create and initialise in one step:
  
  ```
  int myArray[] = {3, 6, 3, 1, 6, 3, 4, 1};
  ```
Iterating through arrays

• *for* loops are useful when dealing with arrays:

```java
for (int i = 0; i < myArray.length; i++)
{
    myArray[i] = getsomevalue();
}
```
Arrays of objects

• So far we have looked at an array of primitive types.
  – integers
  – could also use doubles, floats, characters…
• Often want to have an array of objects
  – Students, Books, Loans ……
• Need to follow 3 steps.
Declaring the array

1. Declare the array
   
   private Student studentList[];
   
   – this declares studentList

2. Create the array
   
   studentList = new Student[10];
   
   – this sets up 10 spaces in memory that can hold references to Student objects

3. Create Student objects and add them to the array:
   
   studentList[0] = new Student("Cathy", "Computing");
JAVA methods
Classes are Object definitions

- OOP - object oriented programming
- code built from objects
- Java these are called classes
- Each class definition is coded in a separate .java file
- Name of the object must match the class/object name
How are simple methods defined?

Every method is defined inside a Java class definition

```java
public class Movie {
    public static int movieRating(int s, int a, int d) {
        return s+a+d;
    }
}

public class Demo {
    public static void main (String argv[]) {
        int script = 6, acting = 9, directing = 8;
        displayRating(script, acting, directing);
    }
    public static void displayRating(int s, int a, int d){
        System.out.print(“The rating of this movie is”);
        System.out.println(Movie.movieRating(s, a, d));
    }
}
```
Main method

• The main method of class Echo starts with the line:
  public static void main (String[] args) throws IOException
• throws IOException is necessary for programs that perform Input/Output.
• It informs the compiler that main performs an input operation that might fail.
• When the program is running and an input operation fails, the computer system is informed of the failure and the program halts.
The three principles of OOP

• Encapsulation
  – Objects hide their functions (methods) and data
    (instance variables)

• Inheritance
  – Each subclass inherits all variables of its superclass

• Polymorphism
  – Interface same despite different data types
Method Signatures

• A method signature specifies:
  – The name of the method
  – The type and name of each parameter
  – The type of the value (or object) returned by the method.
  – \textit{modifiers type name ( parameter list ) [throws exceptions ]}

\begin{verbatim}
public float convertCelsius (float tCelsius ) {}

public boolean setUserInfo ( int i, int j, String name ) throws IndexOutOfBoundsException {}
\end{verbatim}
Public/private

• Methods/data may be declared *public* or *private* meaning they may or may not be accessed by code in other classes …

• Good practice:
  – keep data private
  – keep most methods private

• well-defined interface between classes - helps to eliminate errors
Using objects

• Here, code in one class creates an instance of another class and does something with it …

    Fruit plum=new Fruit();
    int cals;
    cals = plum.total_calories();

• Dot operator allows you to access (public) data/methods inside Fruit class
Constructors

• The line

    plum = new Fruit();

• invokes a constructor method with which you can set the initial data of an object

• You may choose several different type of constructor with different argument lists

    eg Fruit(), Fruit(a) ...
Overloading

• Can have several versions of a method in class with different types/numbers of arguments

  Fruit() {grams=50;}
  Fruit(a,b) { grams=a; cals_per_gram=b;}

• By looking at arguments Java decides which version to use
Thank You