Advanced Programming

Reviewing Basics of Java Programming Language

Java—Why?

- Portable - Write Once, Run Anywhere
- Security has been well thought through
- Robust memory management
- Designed for network programming
- Multi-threaded (multiple simultaneous tasks)
- Dynamic & extensible (loads of libraries)
  - Classes stored in separate files
  - Loaded only when needed

Java Hello World

```
/* This is a hello world example in Java
   that will simply display Hello World
   on the monitor */

public class HelloWorld
{
        public static void main(String args[])
        {
                System.out.println(“Hello World”);
        }
}
```

Comments

```
/* This is a hello world example in Java
 * that will simply display Hello World
 *  on the monitor */
```

- Block Comment at start to describe purpose
  - /* … *comment* …*/
- Line comments used between statements
  - // *comment*

Class

```
public class HelloWorld {
   . . .
}
```

- At least one class per java file
  - Starts with keyword `public class`
  - Followed by class name
- All names have rules to follow
    - Each word in class name starts in uppercase (convention)
    - No punctuation (except underscore) and no spaces
    - Do not start with a number
    - Java file name must be same as class name

Main method

```
public class HelloWorld {
    public static void main(String[ ] args) {
        . . .
    }
}
```

- Java classes are structured into methods
  - Each java application must have one **`main`** method
- Main method always has same *signature*
  - Other methods differ

Statements

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

- Statements are terminated by semicolon
- Statements consist of construct and expression
  - Construct is the command
  - Expression is the data to be enacted upon

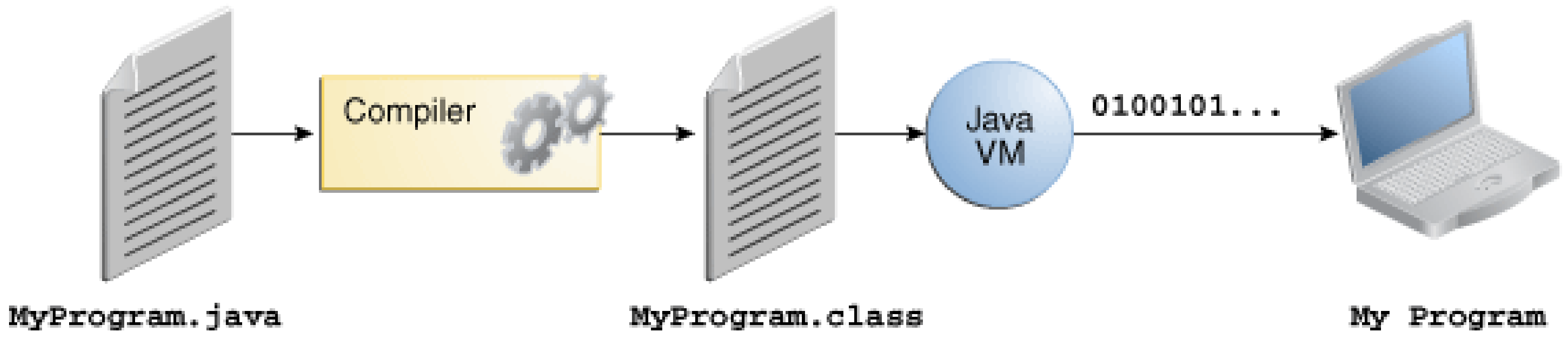
Compiling and Executing Java Programs

- Compilation

javac classname.java

- Execution

java classname



The Java Virtual Machine (JVM)

- Run-time Environment for Java programs.
- The JVM is machine dependent.
- The .class files contain Java bytecodes.
- Provides platform independence: Any platform having a JVM can execute the class files.
- The class files have a defined format that is followed by the Java compilers.
- Just In Time (JIT) compilation tries to increase speed.

Java Primitive Types

- Pre-defined by Java Programming Language and named by its reserved keyword.
- This means that you don't use the new operator to create a primitive variable.
- Declaring primitive variables:

```
float initVal;
int retVal, 2;
double gamma = 1.2;
boolean valueOk = false;
```

Type	Size
byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
char	2 bytes
boolean	1 bit

Basic Mathematical Operators

- `*  /  %  +` – are the mathematical operators
- `*  /  %` have a higher precedence than + or –

```
double val = a + b % d – c * d / b;
```

- Is the same as:

```
Double val = (a + (b % d)) –
                 ((c * d) / b);
```

Assignment Operators

- `=` Assignment operator
- When a calculation involves one variable on both sides we can use an assignment operator

```
 +=     -=      *=    /=    %=
```

- For example if we wish to increase the variable `num` by `10` the full calculation is

```
 num = num + 10;
```

- As only `num` is being used we can apply the `+=` assignment operator

```
  num += 10;
```

Unary Operators

- If an `int` variable is to be increased by `1`, then we can apply the pre/post unary incremental operator
  - `++num` or `num++`
- If an `int` variable is to be decreased by `1`, then we can apply the pre/post unary decremental operator
  - `--num` or `num--`
- We use these operators as part of an statement
  - Pre operator increments/decrements at start of statement
  - Post operator increments/decrements at end of statement

Statements & Blocks

- A simple statement is a command terminated by a semi-colon:

  ```
  x = 2;
  ```

- A block is a compound statement enclosed in curly brackets:

  ```
  {
      x = 2; y = 3;
  }
  ```

- Blocks may contain other blocks

Methods

- A method is a standalone block of code, which
  - Is only run when invoked (by its name)
  - Designed to achieve a set task
  - May accept data when being invoked, via parameter passing
  - May or may not return a result, i.e. return type
- So far we have only written code in the main method
  - But now we will write code in separate methods

Method Format and Examples

- **Format**

```
[modifier] [static] returnType methodName (parameters) {
  //method code
}
```

- No return type example, (no body and no parameters)

```
private void emptyMethod(){
}
```

- Return type example (body, parameter and return line)

```
private static int getPerimeter(int length){
  return 4 * length;
}
```

Using Methods

- A method can be invoked by any code within the same class
  - However the `main` method is always the starting point for the whole program
  - We will often invoke methods from `main`
  - In which case the methods should be marked `static`
- To invoke a method we simply call the name of the method and supply any needed arguments

```
emptyMethod();
```

- If a method returns a value then we can assign the method call to a variable:

```
perimeter = getPerimeter(length);
```

Control Flow Statements

- Normally control flows from top to bottom in a method. Control flow statements break up the flow of execution by employing decision making, looping, and branching, enabling your program to *conditionally* execute particular blocks of code.
- Decision-making statements (if-then, if-then-else, switch)
- Looping statements (for, while, do-while)
- Branching statements (break, continue, return)

If – The Conditional Statement

- The if statement evaluates an expression and if that evaluation is true then the specified action is taken

```
if ( x < 5 ) x = 10;
```

- If the value of x is less than 5, make x equal to 10
- It could have been written:

```
if ( x < 5 )
x = 10;
```

- Or, alternatively:

```
if ( x < 5 ) { x = 10; }
```

Relational Operators

==	Equal
!=	Not equal
>=	Greater than or equal
<=	Less than or equal
>	Greater than
<	Less than

If… else

- The if … else statement evaluates an expression and performs one action if that evaluation is true or a different action if it is false.

```
if (x != oldx) {
   System.out.print("x was changed");
}
else {
   System.out.print("x is unchanged");
}
```

Nested if … else

```
if ( CONDITION1 ) {
  if ( CONDITION2 ) {
    System.out.println("Condition1 and
  Condition2 both are true");
  }
  else {
    System.out.println("Condition1 is true
  and Condition2 is not");
  }
}
else
{
  System.out.println("Condition1 is not
  true");
}
```

else if

- Useful for choosing between alternatives:

```
if ( CONDITION1 ) {
  // execute code block #1
}
else if ( CONDITION2 ) {
  // execute code block #2
}
else {
  // if all previous tests have failed,
  execute code block #3
}
```

The switch Statement

```
switch ( n ) {
  case 1:
   // execute code block #1
   break;
  case 2:
   // execute code block #2
   break;
   default:
   // if all previous tests fail then
   //execute code block #4
   break;
}
```

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

```
n=0
while(n<10) {
  System.out.print( " The value of n is" + n);
  n++;
}
```

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

What is the maximum number of times?

do {… } while loops

```
n=0
do {
   System.out.print( “ The value of n is” + n);
  n++;
} while(n<10);
```

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.

```
for ( int i = 0; i < n, i++ ) {
  if ( CONDITION1) {
   // statements here
   break;
  }
} // program jumps here after break
```

return

- Exits a method with or without a value.
- Discussed earlier in Methods