Session 1

Table of Contents

Intro to Java

What is Java?

1. A programming language
2. Object oriented
3. WORA (Write-Once-Read-Anywhere)
4. Platform Independent

Flavors of Java

1. ME - Mobile Edition,
2. SE - Standard Edition
3. EE - Enterprise Edition

Compilation of a java file

JDK, JRE, JVM, bytecode

JDK = Java Development Kit, provides you with libraries and binaries to develop Java programs
JRE = Java Runtime Environment, provides you with libraries to run Java programs
JVM = Java Virtual Machine, responsible for actually running compiled Java code (bytecode).
The JVM is what makes Java platform independent.
The JVM itself is not platform independent.

When compiling, remember that `javac` requires the **filename** as a parameter.
When running, remember that `java` requires the **name of the main class** as a parameter

To avoid confusion, (and by convention), you should name both the file and the class it contains with the same name, **but this is not strictly compulsory, unless your class is public**. So `HelloWorld.java` would have a class called `HelloWorld`, which would have to have a method called `main` with the correct signature (See Hello World example below.)

```
javac HelloWorld.java
java Hello
```

Naming Conventions

Variables and functions: `camelCase`
Examples: `noOfCats`, `scanner`, `totalSalary`, `getAmount`

Classes: `PascalCase`
Examples: `SampleJavaApplication`, `CodeChefSolution`

Constants: `ALL_CAPS_WITH_UNDERSCORES`
Examples: `PI`, `SECONDS_IN_HOUR`

Hello World

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

`System`, `out`, and `println`
Signature of a function = name of the method + number and type of arguments (return type is not part of the signature in Java)
signature of `main` = `main` + a single argument which is an array of Strings.

If you do not define main correctly, but are close enough, these are the messages you would see **on runtime**:

```
# Changed the return type
Error: Main method must return a value of type void in class HelloWorld, please
define the main method as:
  public static void main(String[] args)

# Removed the static identifier
Error: Main method is not static in class HelloWorld, please define the main method as:
  public static void main(String[] args)

# Removed the public identifier
# Note how it says it couldn't "find" the method at all.
Error: Main method not found in class HelloWorld, please define the main method as:
  public static void main(String[] args)
```

Primitive data types

Primitives in Java (`byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `char`) and their ranges

Primitive Data Type	Size (bytes)	Size (bits)	Range	Range in powers of 2
byte	1 byte	8	-128 to 127	-2^{7} to $2^{7} - 1$
short	2 bytes	16	-32,768 to 32,767	-2^{15} to $2^{15} - 1$
int	4 bytes	32	-2,147,483,648 to 2,147,483, 647	-2^{31} to $2^{31} - 1$
long	8 bytes	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	-2^{63} to $2^{63} - 1$
float	4 bytes	32	~ ±3.40282347E+38F (6-7 significant decimal digits)	-
double	8 bytes	64	~ ±1.79769313486231570E+308 (15 significant decimal digits)	-
char	2 byte	16	0 to 65,536 (unsigned)	0 to $2^{16} - 1$
boolean	not precisely defined*	-	true or false	-

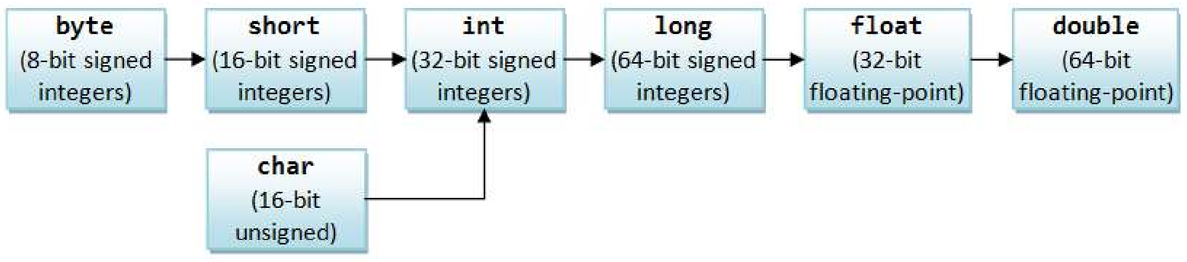
Arithmetic operations: `+, -, *, /, %`
Shorthand operations: `++, --, +=, -=, *=, /=, %=`
Comparative ops: `<, >, <=, >=, ==, !=`
Logical ops: `&&, ||`
Bitwise ops: `&, |, ^, >>, <<`
Short-circuit behaviour
Ternary operator `condition ? value1 : value2`

Basic modular arithmetic rules: These rules are particularly useful when the result of a calculation overflows.

1. `(a + b) % c == ((a % c) + (b % c)) % c`
2. `(a * b) % c == ((a % c) * (b % c)) % c`

Note: The same does not apply for division. Modular division is a slightly more involved process, and you are not expected to know it.

Implicit and explicit casting
Casting from and to different data types


byte (8-bit signed integers)
short (16-bit signed integers)
int (32-bit signed integers)
long (64-bit signed integers)
float (32-bit floating-point)
double (64-bit floating-point)
char (16-bit unsigned)
Orders of Implicit Type-Casting for Primitives

Be careful with shorthand notation (`+=, -=`) because of implicit casting
Be careful with floating point precision

Some real coding

Input using `Scanner` class

```
Scanner sc = new Scanner(System.in);
```

Practice:

1. Take n, print sum of first n numbers
2. Take k, print sum of first k odd numbers
3. Take k, print sum of first k even numbers

Conditionals:

if-else

```
if (<boolean-expression>) {
    // do stuff
}
```

```
if (<boolean-expression>) {
    // do stuff
}
else {
    // do some other stuff
}
```

```
if (<boolean-expression>) {
    // do stuff
}
else {
    if (<some-other-condition>) {
        // do something
    }
    else {
        // do something else
    }
}

// Equivalent
if (<boolean-expression>) {
    // do stuff
}
else if (<some-other-condition>) {
    // do something
}
else {
    // do something else
}
```

Some fun with if conditions.
Are these possible

i > i + 1

i == i + 1

i != i + 0

switch-case

Typically useful when checking specific values on a variable

```
int c = sc.nextInt();
switch(c) {
    case 2: System.out.println("Wow");
            break;
    case 0: int k = 5;
            System.out.println("Look ma, no braces");
            break;
    case 4: System.out.println("I don't even need to be in order");
```

```
            break;
        default:
            System.out.print("You're out of luck");
            break;
    }
```

Remember that switch has fallthrough logic, so once a case matches, it will execute all following cases as well. E.g:

```
    int i = sc.nextInt() % 3;
    switch(i) {
        case 0: System.out.println("I am divisible by 3");
        case 1: System.out.println("I am not divisible by 3");
        case 2: System.out.println("I am not at all divisible by 3");
    }
```

If you were to enter 3 (or any multiple of 3), it will print:

> I am divisible by 3
> I am not divisible by 3
> I am not at all divisible by 3

This is why you need to remember to put the `break;` statements in each case.

You can skip the default case, but it is better to include it.

Practice:

1. https://www.hackerrank.com/challenges/compare-the-triplets

Loops

There are three styles of looping
1. for
2. while
3. do-while

for

```
    for (<initialization>;<condition>;<update>) {
        // do stuff
    }
```

Any/all of the three components of a for loop can be left out. The following are valid for loops:

```
    // Infinite
    for(;;) {
    }

    int i = 0;
    for (;i < 5; i++){
        System.out.println(i);
    }

    // Multiple declarations and updates are also okay!
    for (int i = 0, j = 10; i < j; i++,j--) {
        System.out.println(i);
    }
```

while

```
while (<condition>) {
}
```

Example:

```
while (i < 10) {
    System.out.println(i);
}
```

A common pattern in competitive programming is for the question to have multiple test cases in a single run. In such cases, a while loop is typically useful:

```
// no of test cases
int T = sc.nextInt();

while (T-- != 0) {      // note the post-decrement
    int n = sc.nextInt();
    // ...
}
```

do-while

```
do {
    // some stuff
}
while (<condition>);    // <- note the semicolon after the end
```

Practice

1. https://www.hackerrank.com/challenges/simple-array-sum
2. https://www.hackerrank.com/challenges/filling-jars
3. https://www.hackerrank.com/challenges/mini-max-sum

Home Assignments

HackerRank

1. https://www.hackerrank.com/challenges/handshake
2. https://www.hackerrank.com/challenges/summing-the-n-series
3. https://www.hackerrank.com/challenges/staircase
4. https://www.hackerrank.com/challenges/kangaroo
5. https://www.hackerrank.com/challenges/utopian-tree
6. https://www.hackerrank.com/challenges/strange-grid
7. https://www.hackerrank.com/challenges/angry-professor
8. https://www.hackerrank.com/challenges/rectangular-game

Miscellaneous

1. Write a calculator program. Specifications:

    i. Display a menu to the user. Menu should have the following options:
        a. 1 = Addition, 2. Subtraction, 3 = Multiplication, 4 = Division, 5 = Exit

    ii. Accept an integer choice. If the choice is either 1, 2, 3 or 4, accept two **real numbers** from the user. (Hint: Search what real numbers are on Google, and decide on an appropriate data type for them.)

    iii. Perform the appropriate action on the two numbers.

iv. Display the output.
v. If the user input an incorrect choice, inform him/her. Do **not** exit.
vi. Continue steps 1 to 5, till the user chooses to exit.

Some interesting cases:
1. What do you expect will happen when you choose option 4, and the second operand is 0? What actually happens? What happens when you change the data type to plain integers?
2. Does this help you answer an earlier puzzle?
3. Pick option 2, enter the first operand as 2.1, and the second as 1.2. What output do you expect? What do you actually get? Why?