Chapitre 4

Working with Basic Classes and Variables

Primitive Types

A primitive is the most basic form of data in a Java program. The Java language has eight primitives. All primitive types are represented by lowercase letters and are divided into the following types:

- Integer
- Floating point
- Character
- Boolean

Integers(1)

This primitives represent integers in a decimal, hexadecimal or octal form.

long Recommended when the number is very large, capable of storing 64-bit.

int The most commonly used integer type, capable of storing 32-bit.

short Numeric value capable of storing 16-bit.

byte Used to represent small numbers in the range of -128 to 127, capable of storing 8 bits.

Integers (2)

- Examples:

byte : 0, 1, 2

short : 1, 20, 100

int : 10, 100, 2000000

long : 1000, 21234300000, 56098001258

Floating Point Numbers

A floating point number is a primitive that is used to store decimal values.

- float: Is a floating point number capable of storing 32-bit.
- double: It is like a float and stores up to a 64-bit floating-point number.
- Examples:

- float: 1.99, 13.14, 100.45
- double: 1500.27, 409673.90, 129259246.062419

Characters

char : Is used to store a single 16-bit Unicode character and requires 16 bits of memory.

To represents a single character, including letters, numbers, special characters and others, The range of a char corresponds to the minimum and maximum as defined by the Unicode specification '\u0000' (or 0) and '\uffff' (or 65,535 inclusive), respectively.

- Examples:

char: a,b,c,4,$,&

Booleans

boolean: Is used to store a logical value of true or false. They store a one-bit value and will default to false.

- Examples:

boolean: true,false

Integer Types (byte, short, int, long) (1)

- Integer types basically handle positive or negative integers. The values can be represented in octal, decimal or hexadecimal bases.
- **Octal Integer Literal**

  Any value written using numbers from 0 to 7 starting with a value 0 is the octal base, for example:

  short s = 010 // is equivalent to the value 8 in decimal.
  int i = 025 // is equivalent to the value 21 in decimal.

Integer Types (byte, short, int, long) (2)

- **Decimal Integer Literal**
  Any value written using numbers from 0 to 9 is a decimal value, this is the most common type of representation, for example:

```
int i = 9;
long b = 9871342132;
```

Integer Types (byte, short, int, long) (3)

- **Hexadecimal Interger Literal**

  Any value written using numbers **0** through **9** and **A** to **F** starting with 0x or 0X is a hex value, for example:

  long a = 0xCAFE // is equivalent to the value 51966 in decimal

  int b = decimal 0X14a3;
  // Equivalent to the value 5283 in decimal

Integer Types (byte, short, int, long)(4)

- When a number is larger than the size of an int type we need to specify the type as a long, for that we should add the letter l or L after its value, for example:

```
long l = 0Xcafel;
long b = 0752L;
long c = 987654321L;
```

Floating point types (float and double)

- The floating-point types are **float** and **double**, which are conceptually associated with the single-precision 32-bit and double-precision 64-bit format. All floating point numbers are of type double by default.

```
float f = 10.99f;
double b = 10.3D;
double C = 10.3;
/ / Compile error because the default value is double.
float c = 1.99;
```

- Floating decimal point numbers are written with one period to show the decimal point (.). A negative number begins with a single -.

Character type (char)

- The type character, as its name implies, serves to represent a value of this type. For example:

```
char a = 'a' // Start with an unusual character.
char a = 97 // Start with ASCII code.
char u = ‘\u0025' // Equivalent to '%' character
```

- The characters can be represented by numbers and have the same size as an short attribute.
- The Unicode is in hexadecimal format, so the previous example '0025 'equals 37 in decimal base.

Sun ORACLE

Java Is Strongly Typed

- Java is a strongly typed language. Variables must be declared as a type, and any value that is stored must be compatible with this type.
- It is possible to cast a variable to a different data type. If incompatible types are cast, an exception will be thrown.
- The following is an example of casting variables and data:

```
float floatNum = 1.5f;
int wasFloat = (int) floatNum;
```

- The variable wasFloat would be equal to 1 since the .5 would be truncated to make the data compatible.

Casting (1)

from\to	byte	short	char	int	long	float	double
byte		implicit	char	Implicit	implicit	implicit	implicit
short	byte		char	implicit	implicit	implicit	implicit
char	byte	short		implicit	implicit	implicit	implicit
int	byte	short	char		implicit	implicit	implicit
long	byte	short	char	int		implicit	implicit
float	byte	short	char	int	long		implicit
double	byte	short	char	int	long	float	

Casting (2)

- To make a casting, we just signal the type to which to convert the between parentheses as follows:

```
float f = (float) 5.0;
```

Conversion from double to float.

```
int b = (int) 5.1987;
```

Convert double to int.

```
float c = 100;
```

Conversion from int to float is implicit, does not need casting.

```
int d = 'd';
```

Converting char to int is implicit, does not need casting.

Sun ORACLE

Enumerations

Enumerations are a special data type in Java that allows for a variable to be set to predefined constants. The variable must equal one of the values that have been predefined for it.

- Examples:

```
enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }
Suit cardSuit;
cardSuit = Suit.CLUBS;
if(cardSuit == Suit.CLUBS){
System.out.println("The suit of this card is clubs.");
}
```

Enumeration (1)

- Enums can be declared as their own class, or enclosed in another class, and that the syntax for accessing an enum's members depends on where the enum was declared. They can’t be declared into methods.
- Enumeration are an alternative to use constant values.
- Problems when using constant:
  - The constants do not provide security.
  - Fragile model.
  - Their values are slightly informative.

Enumeration

- Example of creating enum:

```
public enum ConceptEnum {
BAD,
GOOD,
FAIR,
EXCELLENT;
public void calculateApproval ( ConceptEnum concept){
   if (concept==ConceptEnum.EXCELLENT) {
      System.out.println ("Passed with flying colors!");
   } else if (concept == FAIR){
      System.out.println ("Regular");
   }else if(concept== GOOD){
    System.out.println ("Accepted");
     }else if (concept == BAD){
     System.out.println("Failed");}
       }
       }
```

Sun ORACLE

Enumeration

- Because an enum really is a special kind of class, you can do more than just list the enumerated constant values. You can add **constructors**, **instance variables**, **methods**, and something really strange known as a ***constant specific class body***.

```
enum CoffeeSize {
// 8, 10 & 16 are passed to the constructor
BIG(8), HUGE(10), OVERWHELMING(16);
CoffeeSize(int ounces) { // constructor
this.ounces = ounces;
}
private int ounces; // an instance variable
public int getOunces() {
return ounces;
}
}
```

Enumeration (5)

- Test example :

```
class Coffee {
CoffeeSize size; // each instance of Coffee has an enum
public static void main(String[] args) {
Coffee drink1 = new Coffee();
drink1.size = CoffeeSize.BIG;
Coffee drink2 = new Coffee();
drink2.size = CoffeeSize.OVERWHELMING;
System.out.println(drink1.size.getOunces()); // prints 8
for(CoffeeSize cs: CoffeeSize.values())
System.out.println(cs + " " + cs.getOunces());
} }
```

Which produces:

```
8
BIG 8
HUGE 10
OVERWHELMING 16
```

Sun ORACLE

Arrays

- An array is a series of variables of the same type that can be accessed by an index. Arrays are useful when you have a set of data that will all be used in a similar way. Arrays can be made from primitive data types or objects.
- Example:

```
int[] testScore = new int[3];
testScore[0] = 98;
testScore[1] = 100;
testScore[2] = 72;
int shannonsTestScore = testScore[1];
```

Array

A vector is composed of three parts:

- declaration

```
int [] values ; // or int values [];
String [] names; // or String names [];
```

- creation

```
int [] values = new int [100];
String [] names = new String [50];
```

- Initialization

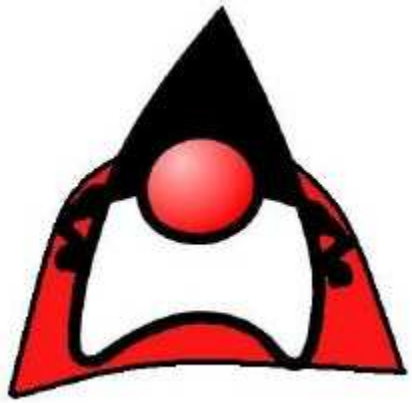
```
values [0] = 5;
names [10] = "Rafael";
```

- We can also create and initialize the array as follows:

```
int [] values = {2, 5, 4, 8, 5};
```

Object (1)

Characteristics+ behavior = Object



Sun ORACLE

Object (2)

- An object is a representation of a set of characteristics and behavior to represent something of the real world.
- We represent people such as objects, where each person has their own characteristics such as name, date of birth, passport, etc .
- We can also represent the behavior that an object has as the actions that a a person can do, such as walking, thinking, speaking and others.
- In OO characteristics are called attributes and actions are called methods, where all objects of the same type have the same methods, but each one may have its attributes with different values (state, instance variables).

Object (3)

From a model or a template defining the attributes and the methods of objects ,we can create multiple objects each one with a different state:

Model

Objects with different states

Java Naming Conventions

- A class should be named with the first letter capitalized, along with each sequential word in the name.

```
class SportCar {…}
class BaseballPlayer {…}
class Channel {…}
```

- A variable should be named with the first letter being lowercase, and with each sequential word in the name beginning with a capital.

```
int milesPerGallon;
float price;
int i;
Car raceCar;
```

Any Questions ?