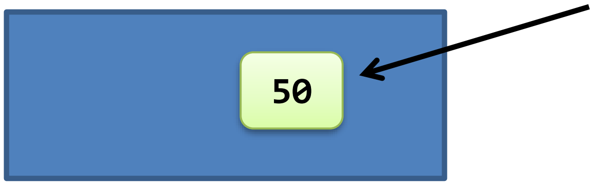
CS 2100 – Java Review [02]

Data Structures and Algorithms 1

Java Data Types

Variable

A variable is simply a name associated with a reserved area allocated in memory. Think of it as a named box in memory that contains data.



Green box: *Reserved Area, called "num" (value=50)*

RAM:

```
int num = 50; // Here "num" is the variable, storing the numerical value 50.
```

Types of Variables

There are three main types of variables in Java.

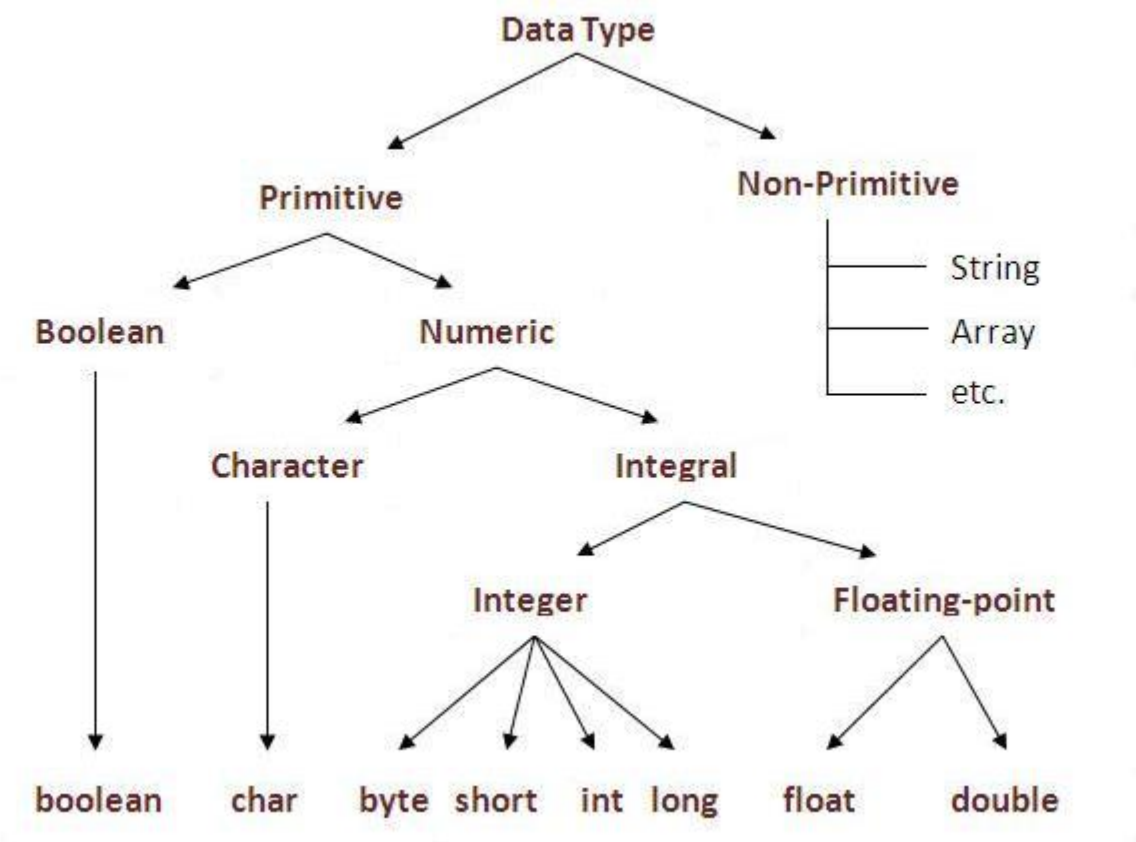
- **LOCAL VARIABLES** – Similar to how an object stores its state in fields/attributes, a method will often store its temporary state in local variables. The syntax for declaring a field is the same for declaring a local variable (more on declaration below.) There is no special keyword necessary when declaring a local variable. A local variable is identified entirely from the *location* in which the variable is declared – which is between the opening and closing braces ("**{**" and "**}**")of a method. Local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.
- **INSTANCE VARIABLES (NON-STATIC FIELDS)** – Technically, objects store their individual states in "non-static fields", that is, fields declared without the `static` keyword. Non-static fields are also known as *instance variables* because their values are unique to each *instance* of a class (to each object, in other words). They are declared inside the class but outside any method. Example: the `computingID` of one student is independent from the `computingID` of another.
- **CLASS VARIABLES (STATIC FIELDS)** – A *class variable* is any field declared with the `static` modifier; this tells the compiler that there is exactly one copy of this variable in existence; regardless of how many times the class has been instantiated. A static variable cannot be local. Example: A field defining the number of gears for a particular kind of bicycle could be marked as `static` since conceptually the same number of gears will apply to all instances. The code `static int numGears = 6;` would create such a static field.

Data Types

The Java programming language is *statically-typed*, which means that all variables must first be declared before they can be used. This involves stating the variable's *type* and *name*, such as:

```
int radius = 5;
```

The above statement tells the program that a field named "radius" exists, holds numerical data (`int` stands for *integer* value), and has an initial value of "5." A variable's data type determines the values it may contain, plus the operations that may be performed on it. In addition to `int`, the Java programming language supports seven other primitive data types. Other than primitive data types, Java supports what are known as reference/object data types.



Primitive Data Types

A primitive type is predefined by the language and is named by a reserved keyword. There are eight primitive data types supported by Java:

- **BYTE:**
  - Byte data type is an 8-bit signed two's complement integer.
  - Minimum value is -128 (-2^7); Maximum value is 127 (inclusive)(2^7 -1).
  - Default value is 0.
  - Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
  - Example: `byte a = 100;  byte b = -50;`

- **SHORT:**
    - Short data type is a 16-bit signed two's complement integer.
    - Minimum value is -32,768 (-2^{15}); Maximum value is 32,767 (inclusive) (2^{15} -1).
    - Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int.
    - Default value is 0.
    - Example: `short s = 10000;  short r = -20000;`
- **INT:**
    - Int data type is a 32-bit signed two's complement integer.
    - Minimum value is - 2,147,483,648.(-2^{31}); Maximum value is 2,147,483,647(inclusive).(2^{31} -1).
    - Int is generally used as the default data type for integral values unless there is a concern about memory.
    - The default value is 0.
    - Example: `int a = 100000;  int b = -200000;`
- **LONG:**
    - Long data type is a 64-bit signed two's complement integer.
    - Minimum value is -9,223,372,036,854,775,808 (-2^{63}).
    - Maximum value is 9,223,372,036,854,775,807 (inclusive) (2^{63} -1).
    - This type is used when a wider range than int is needed.
    - Default value is 0L.
    - Example: `long a = 100000L;  long b = -200000L;`
- **FLOAT:**
    - Float data type is a single-precision 32-bit IEEE 754 floating point. $\pm 3.4\text{x}10^{38}$
    - Float is mainly used to save memory in large arrays of floating point numbers.
    - Default value is 0.0f.
    - Float data type is never used for precise values such as currency.
    - Example: `float f1 = 234.5f;`
- **DOUBLE:**
    - double data type is a double-precision 64-bit IEEE 754 floating point. $\pm 1.7\text{x}10^{308}$
    - This data type is generally used as the <u>default</u> data type for decimal values.
    - Double data type should never be used for precise values such as currency.
    - Default value is 0.0d.
    - Example: `double d1 = 123.4;`
- **BOOLEAN:**
    - boolean data type represents one bit of information.
    - There are only two possible values: true and false.
    - This data type is used for simple flags that track true/false conditions.
    - Default value is false.
    - Example: `boolean found = true;`

- **CHAR:**
    - char data type is a single 16-bit Unicode character.
    - Minimum value is '\u0000' (or 0).
    - Maximum value is '\uffff' (or 65,535 inclusive).
    - Char data type is used to store any character.
    - Example: `char letterA ='A';`

Other Special (non-primitive) Data Types and Reference Data Types

In addition to the eight primitive data types listed above, the Java programming language also provides *special support* for character strings via the `java.lang.String` class. Enclosing your character string within double quotes will automatically create a new `String` object.
Example: `String s = "this is a string";`.
`String` objects are *immutable*, which means that once created, their values cannot be changed. The String class is not a primitive data type despite the special support given to it by the language.

Reference variables are created using defined *constructors* of the classes (a discussion on constructors will come later). They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example: Employee, Cat, Student, etc. More information on reference data types:

- Class objects, and various types of *array* variables come under reference data type.
- Default value of any reference variable is **null**.
- A reference variable can be used to refer to any object of the declared type or any compatible type.
- Example: `Animal animal = new Animal("giraffe");`
  (This line of code will become clearer after a discussion on declaration of objects and class constructors)

Default Values

Data Type	**Default Value (for fields)**	**Size**
byte	0	1 byte
short	0	2 bytes
int	0	4 bytes
long	0L	8 bytes
float	0.0f	4 bytes
double	0.0d	8 bytes
char	'\u0000'	2 bytes
String (or any object)	null	/
boolean	false	1 bit

It's not always necessary to assign a value when a field is declared. Fields that are declared but not initialized will be set to *a reasonable default by the compiler*. Generally speaking, this default will be zero or null, depending on the data type. Relying on such default values, however, is generally considered bad programming style.

Java Literals

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation. Literals can be assigned to any primitive type variable. For example:

- `byte a = 68;`
- `char capitalC = 'C';`
- `short s = 10000;`
- `boolean result = true;`
- `int i = 100000;`
- `int decimal = 100; // base 10`
- `int octal = 0144; // octal (base 8) – prefix 0`
- `int hexa =  0x64; // hexadecimal (base 16) – prefix 0x`
- `double d1 = 123.4;`
- `double d2 = 1.234e2; // same value as d1, but in scientific notation`
- `float f1 = 123.4f;`
- `String s1 = “Hello World”; // String literal`
- `String s2 = “two\nlines”; // notice the “escape sequence”  \n`
- `String s3 = "\"This is in quotes\""; // notice the “escape sequences”  \"`
- `char x = ‘\u0001’; // char types of literals can contain any Unicode character`
- `String z = “\u0001”; // String types also`

Casting

Before a value can be stored in a variable, the value’s data type must be compatible with the variable’s data type. Java performs some conversions between data types automatically, but does not automatically perform any conversion that can result in the loss of data. Java also follows a set of rules when evaluating arithmetic expressions containing mixed data types.

Primitive data type ranking (Highest to lowest)

`double, float, long, int, short, byte`

In assignment statements where values of lower-ranked data types are stored in variables of higher-ranked data types, Java automatically converts the lower-ranked value to the higher ranked type.

For example:

- `double d1;`
- `int x = 123;`
- `d1 = x;`

Works just fine, while:

- `double d1 = 123.4;`
- `int x;`
- `x = d1;`

Will cause an error. In order for something like this to work we need to *tell* Java that we understand we are *losing precision*. This is called **casting**. Most of the time when you cast you will be going from a double to an int. For example:

- `double d1 = 123.7;`
- `int x;`
- `x = (int)d1;`

After these 3 lines are executed x will contain the value 123, notice that it will truncate the number (it does not round).

Integer Division

When both operands of the division operator are integers, the operator will perform *integer division*. This means the result of the division will be an *integer* as well. If there is a remainder it will be lost. For example:

- `double d1;`
- `d1 = 5/4;`

What value will be stored in d1? Because both 5 and 4 are int literals Java will perform integer division and store 1 in d1 not the expected 1.25. It doesn't matter that d1 is a double above because Java removes the fraction part *before* the assignment is made. In order for the above assignment to work as expected we need to change either the 5 or 4 or both to a *floating point* number, for example:

- `double d1;`
- `d1 = 5.0/4;`

or

- `double d1;`
- `d1 = 5/4.0;`

or

- `double d1;`
- `d1 = 5.0/4.0;`

Will all assign 1.25 to d1. The reason the first two work is that Java converts the int value to a double (see above). Note also that we can cast to avoid integer division as well, for example:

- `int x = 10, y = 4; // notice the use of the` *`comma`* `to indicate x & y are ints`
- `double d1;`
- `d1 = (double)x / y;`

or

- `int x = 10, y = 4;`
- `double d1 = x / (double)y;`

Will store 2.5 in d1. Note that the following will not store 2.5 in d1.

- `Int x = 10, y = 4;`
- `double d1;`
- `d1 = (double)(x/y)`

as Java will do integer division on the x and y before casting.

Escape Sequences

Java language supports few special escape sequences for String and char literals. They are:

Notation	Character represented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Single quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)