**Reference Type**

Java data type can be classified into two types: value type and reference type.

**Value type:**

In Java, value type represents that value of the data is the data itself. For example: the integer 8 means the value is 8. Its value represents the data itself. Value type is a basic data type that we have learned in the preceding part. Basic data types are directly stored in memory stack of the memory. Value of the data is stored in the stack space. In Java language, there are 8 data types which belong to such a storage model.

**Concept of reference type:**

Reference type is a type defined through class. All classes except 8 data types are reference type. Reference type inherits from Object class (also a reference type), in which data is stored in the memory model of the objects stored in Java. Java memory stack and heap are used to store these types of data. To put it simply, “reference” is stored on the memory stack orderly, while value of objects is stored on memory heap.

*Main difference between reference type and value type: Basic data types are assigned to stacks, while reference types are assigned to heaps (the concept of stack and heap in java is required.)*

**Passing:**

**Pass by value:**

Assignment of basic data types belongs to passing by value. Simply variable value is passed in “passing by value”. It passes backup of original parameters. After passing by value, actual parameter passes value to the formal parameter and change of the formal parameter does not affect the actual parameter.

**Pass by reference**

An address is passed in passing by reference. Change of formal parameter will change the value of actual parameter variable.

**Type conversion**

Type conversion happens frequently in Java. It is a very important concept, which contains two aspects:

**Basic type conversion:**

Type conversion happens mainly in three circumstances, i.e. assignment, method calling and arithmetical operation.

1.*Assignment and method calling* conversion rules: Automatic conversion from low type to high type; cast from high type to low type.

* Boolean and other basic data types cannot be converted with each other.
* byte can be converted into short, int, long, float and double；
* short can be converted into int, long, float and double；
* char can be converted into int, long, float and double；
* int can be converted into long, float and double；
* long can be converted into float and double；
* float can be converted into double

1. Type conversion in *arithmetical operation*:

* Convert to high data type first and then involve in operation. The result is also the highest data type;
* byte short char operation will be converted into Int；
* If one of operands is double, the other operand is converted into double first before being put into the arithmetical operation.
* Where the two operands are not double, if one of them is float, the other one is converted into float first before being put into the arithmetical operation.
* Where the two operands are not double or float，if one of them is long, the other one is converted into long first before being put into the arithmetical operation.
* Where the two operands are not double, float or long, both of them are converted into int before being put into the arithmetical operation.

Special Cases:

* If abbreviated operations, e.g. += and \*=, are used, the system will automatically cast the operation result into the type of the target variable.
* Where operators are auto increment operator (++) or auto decrement operator (--), if the operand is byte, short or char, the type does not change.

**Reference type conversion:**

1. Basic type and wrapper class counterpart can be converted with each other automatically. This is the principle of automatic packing and unpacking (it will be discussed in the following part).
2. Subclass can be converted into super-class or interface class directly: the subclass becomes a super-class. Please find examples in the preceding sections.
3. Super-class must be casted into a subclass; and if it is discovered that it is not the corresponding object during running, ClassCastException running abnormality will be thrown.

**Automatic Packing and Unpacking Mechanism**

Automatic packing and unpacking is introduced in Java 1.5. Its purpose is to convert primitive type value into corresponding object. Automatic packing and unpacking mechanism enables us to use primitive type or object type more easily and directly in variable assignment or method calling in Java.

**What is automatic packing and unpacking?**

Automatic packing is that Java automatically converts primitive type value into corresponding object, e.g. from int variable to integer object. This process is called packing. The process of converting integer object into int type value is called unpacking. As packing and unpacking are automatic, they are called automatic packing and unpacking. Wrapper class counterpart of primitive type byte, short, char, int, long, float, double and Boolean are Byte, Short, Character, Integer, Long, Float, Double, Boolean.

**Essential points of automatic packing and unpacking**

* During automatic packing, the compiler calls valueOf to convert primitive type value into the object. During automatic unpacking, the compiler calls such methods as intValue(),doubleValue() to convert the object into primitive type value.
* Automatic packing is to convert the Boolean value into Boolean object, byte value into Byte object, char into Character object, float value into Float object, int into Integer, long into Long and short into Short. Automatic unpacking is a contrary operation.

**Related Reference Links:**

1. [Difference between a Value Type and a Reference Type](http://net-informations.com/faq/general/valuetype-referencetype.htm)
2. [Reference Types](https://www.oreilly.com/library/view/java-8-pocket/9781491901083/ch04.html)
3. [Automatic Packing and Unpacking in Java](http://www.codeceo.com/article/java-auto-pack-unpack.html)

Reading finish