Definitions and concepts

Some High - lights of

Java

By: Hamid Mosavi-Porasl

Lexical Tokens (or just tokens): Tokens are the building blocks for more complex constructs. Identifiers, operators and special characters are all examples of tokens that can be used to build high-level constructs like expressions, statements, methods and classes.

Method automatic variables means local variables. A variable stores values of data types.

Primitive types

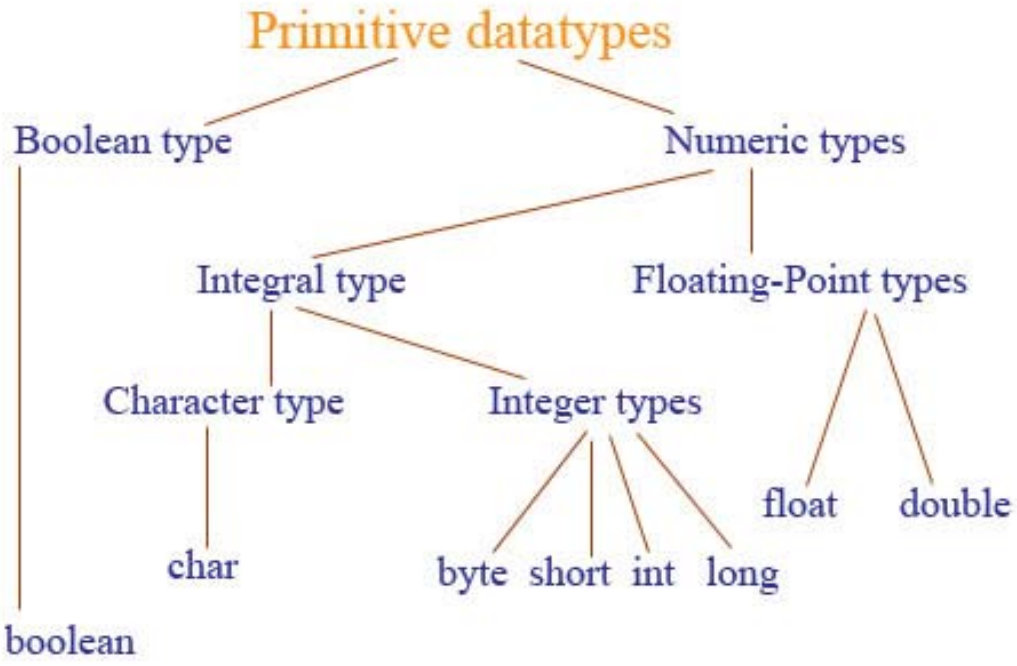

Figure 1 Primitive datatypes

- An ***Integer*** could be a datatype in byte (8 bits), short(16 bits), int (32 bits) or long(64 bits).
- A ***char*** is 16 bits
- A ***float*** is 32 bits long
- A ***double*** is 64 bits long

Wrapper classes are defined in package: java.lang

Datatype	Width	Minimum/Maximum	Wrapper Class

		value	
boolean	N.A.	true, false (no ordering)	Boolean
byte	8	-2^7, 2^7-1	Byte
short	16	-2^{15}, $2^{15}-1$	Short
char	16	0x0,0xffff	Character
int	32	-2^{31}, $2^{31}-1$	Integer
long	64	-2^{63}, $2^{63}-1$	Long
float	32	+,-.40129846432e-45 +,-3.4028234663e+38	Float
double	64	+,-4.9406564e-324 +,-1.7976931e+308	Double

Default values for datatypes

Datataype	***Default value***
boolean	false
char	'\u0000'
Integer(byte,short,int,long)	0
Floating-point(float, double)	+0.0F or +0.00
Object reference	null

Initializing

- local variables are ***not*** initialized when they are instantiated at method invocation.
- Static variables and Instance variables are initialized to default values when the class is instantiated, if they are not explicitly initiated.

Valid java files

- an empty file is a valid java file

Casting

- Type conversions involving casts are called ***explicit conversions***. In certain contexts, the compiler performs ***implicit conversations***, for examples when a char is assigned to an int.

- ***Narrowing and widening Conversations***: For the primitive datatypes, a value of narrower datatype can be converted to a value of broader datatype without loss of information. This is called a ***widening primitive conversion***.
- Conversion from a broader datatype to a narrower datatype is called a ***narrowing primitive conversation***.

The conversations are transitive. For example, an int can be directly converted o a double without first having to convert it to a long and a float.

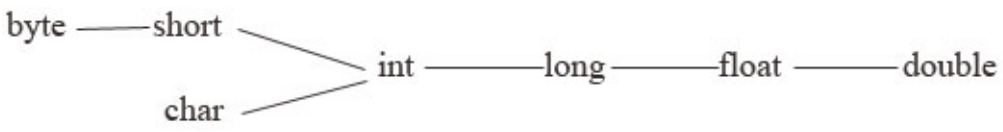

Figure 2 Widening and narrowing conversions

Parameter Passing

Objects communicate by passing messages. A message is implemented as a method call to invoke a particular method on an object, i.e. *objRef.doIt(time,place)*

Static methods can be invoked on classes in Java, e.g. *java.lang.Mat.abs(-1)*

Actual parameters are parameters passed to the method when the method is invoked by a method call, and can vary from call to call. The parameter passing strategy in Java is ***call-by-value*** and not call-by-reference, regardless of the type of the parameter.

Formal parameters are parameters are parameters defined in the method definition and are local to the method.

Final parameter: a formal parameter can be declared with the keyword final preceding the parameter declaration in the method definition. A final parameter is also known as a *blank final variable*, i.e. it is blank (Uninitialized) until a value is assigned to it at method invocation.

Method Overloading

Method overloading allows a method with the same name but different parameters, thus with different signatures, to have different implementation and return values of different types.

Abstract classes

Any class can be specified with the keyword abstract to indicate that it cannot be instantiated. A class might choose to do this if the abstraction it represents is so general that it has to be specialized in order to be practical use.

A class that has an abstract method must be declared abstract.

Final classes

A final class can't be extended. In other words, its behavior can't be changed by subclassing. Only a class whose definition is complete can be specified to be final. A final method cannot be *abstract*.

Final members

A final variable is a constant, and its value cannot be changed once it's initialized. Variables defined in an interface are implicitly final.

Static Members: Static variables also known as ***class variables***. Only exist in the class they are defined. Static methods are also known as ***class methods***. A static method in a class can directly access other statistic members in the class. It cannot access instance members of the class, as there is no object being operated on when a static method is invoked. A typical static method might perform some task on behalf of the whole class and or for objects of the class.

Synchronized methods: One thread at a time can execute the method, then the method can be specified as synchronized in the method definition.

Native methods: **JNI** (Java Native Interface) is a special API, which allows java methods to invoke native methods implemented in C.

Transient Variables: Objects can be stored, using serialization. Such objects are said to be *persistent*. Variable can be specified as transient indicating that its value need not be saved when objects of its class are put in persistent storage. E.g.

```
Class Experiment implements Serializable{
    // the value of current  temperature will not persist
    transient int currentTemperature;  // Transient value
    double mass;    //Persistent value
}
```

Volatile Variables: During execution, threads might cache the values of member variables for efficiency reasons. Since threads can share a variable, it is then vital that reading and writing of the value in the copies and the master variable don't result in any inconsistencies. As this variable's value could be changed unexpectedly, the *volatile* modifier can be used to inform the compiler that it should not attempt to perform optimizations on the variable. Declaring the variable as ***volatile*** ensures that a write operation will always return the correct current value.

```
Class VitalControl{
    //….
  Volatile long clockReading;
    //Two successive reads might give different results.
}
```

Flow control Statements

- Simple if statement
- If statement
- Switch statement

Iteration statements

- While
- Do-while
- For

Transfer statements

- Break

- Continue: The continue statement can be used in a “for” or a “do-while” loop to prematurely stop the current iteration of the loop body and to proceed with the next iteration, if possible.

- Return
- Try-catch-finally

Object oriented Programming Concepts

- Inheritance
    1. *Inheriting from the superclass*, call of superclass methods are possible
    2. *Extending the superclass*, Extending supperclass by additional methods

    3. *Upcasting*: A subclass reference can be assigned to a superclass reference, because a subclass object ca be used where a superclass object can be used. This is called *upcasting*.

    4. *Downcasting*: Casting of a superclass reference to a subclass type is called downcasting, e.g. stringRef =(String) objRef;

- Method overloading: Method with same signature can be defined with changed number of parameters and types.
- Polymorphism and dynamic Method Binding**:** The ability of a superclass reference to donate objects of its own class and its subclasses at runtime is called Polymorphism.

Method Overriding and Variable shadowing

Under certain circumstances, a subclass may override non-static methods inherited from the super class. The following aspects about method overriding should be noted:

- The new method definition must have the same method signature and same return type.
- The new method, in addition, cannot “narrow” the accessibility of the method, but it can “widen” it.
- The new method definition in the subclass can only specify all or none, or a subset of exception classes.

- Whether parameters in the overriding method should be final is at the discretion of the subclass. A method's signature doesn't encompass the final modifier of parameters, only their types and order.
- A subclass cannot override variable members of the super class, but it can *shadow* them.

Interfaces

Java provides interfaces, which not only allow new type names to be introduced and used polymorphically, but also permit *multiple interface inheritance*. The body of the interface is usually a list of method prototypes. An interface is abstracted by definition and therefore cannot be initiated. It should also not be declared as abstract.

A class can choose to implement only some of the methods of its interface, i.e. given a partial implementation of its interfaces. The class must then be declared as abstract. Note that *interface methods cannot be declared statistic, while variables in interfaces are always static and final.*

Interfaces don't contain any implementations. An interface can extend any number of other interfaces and can be extended by any number of other interfaces. Variables in interfaces are always static and final.

Extending Interfaces

An interface can extend other interfaces, using the "extends" clause. Unlike the linear implementation hierarchy involving classes, which always has the object class as the simple root, multiple inheritance of interfaces can result in an inheritance which has multiple roots designed by different interfaces.

Super types

Interfaces define new types. Although interfaces cannot be initiated, variables of an interface type can be declared. If a class implements an interface, then references to objects of this class and its sub classes can be assigned to a variable of this interface type.

The interface that a class implements and the classes it extends directly or indirectly are called **supertypes**. A supertype is thus a reference type.

Constants in Interface

An interface can also define constants. Such constants are considered to be public, static and final regardless of whether these modifiers are specified. An interface constant can be accessed by any client (a class or interface) using its fully qualified name; regardless of whether the client extends or implements its interface. However, if a client is a class that implements this interface or an interface that extends this interface, then the client can also access such constants directly without using the dot (.) notation.