Java 17 study guide Notes

Chapter 1. Building Blocks

1. The method name and the parameter types are called as method signature

2. The method signature for method

```
    public void setName(String newName) {
          this.name = newName;
    }
```

   is **setName(String)**

3. top-level type is a class (in this chapter)

4. nested class is allowed as long on atmost only one class is public

5. the top level type of a class does not need to be public class ClassName is also permitted

```
    class ClassNotPublic {

          public static void main(String args[]) {
                System.out.println("hello world");

          }
    }
```

Runs just fine

6. The import statement does not bring the classes inside the child packages. It only brings in the classes that are directly under the package.

7. we do not need to import the classes present in the same package as java Automatically looks in to the current package. So we do not need to import the classes present in the same package.

Data Types

boolean	default value is false
byte	default value is 0
short	default value is 0
int	default value is 0
long	default value is 0L
float	default value is 0.0f
double	default value is 0.0
char	default value is \u0000

Text Block (“”” .... “””)

a text block requires line break after “”” to work
example:

this one will not work

```
String name = “””joe”””;
```

but this one will work

```
String name = “””
                joe
```

“””;

Identifiers

java identifiers is the name of variable, method, class, interface, or package.

Identifier rules:

1. must begin with letter, a currency symbol like $,rupee or _
2. can include numbers but cannot start with number like 1, 2, etc..
3. single underscore is not allowed as an identifiers but we can have single currency symbol as a valid identifier

 example:
```
  int _ = 5; //invalid
  int ₹ = 10 //valid
```
4. cannot use reserved words like class, for, public, private etc... as identifier
5. cannot use literal values like true, false, null

Variable declaration

1. cannot declare variables of different types in same line

   example:
```
        int number, String name; // invalid
        int number, int number2; // invalid
        int number; String name;// valid as ; ends one statement. But not recommended as this is not the convention we follow.
```

2. Local variable is a variable that is declared within a constructor, method, or initializer block.
3. final keyword can be applied to local variable and is equal to constant. ie.. the value of a final variable cannot be altered.

	Example:
```
        final int sum = 10;
        int number = 5;
        sum = sum + number; // invalid as the sum variable is final so it cannot be altered.
```

	but for final reference like array we can still be able to change the value of the individual array elements but we cannot be able to change the reference itself

	Example:
```
        final int [] favNumbers = new int[10];
        favNumbers[0] = 10; // valid
        favNumbers [1] =30; // valid
        favNumbers = null; // will be invalid as the reference of the array is changed
```

4. if a LOCAL variable is used before it is initialized it will throw an error. For both primitive types and reference type.
5. Variables passed on as method or constructor parameters also needs to be initialized. Else they will not compile.

	Example:
```
          public void findAnswer( boolean check){
}
        public void checkAnswer(){
        boolean value;
        findAnswer(value);// Do not compile as the value “value” is not initialized.
}
```

5. INSTANCE and CLASS variables do not need to be initialized in order to use them.

These are given default values like 0 for numeric and null for objects and ‘\u0000’ (NUL) in case of char.

6. Instance variable are the normal variables declared inside a class (not inside functions or initializer blocks inside a class) and needs an object to access that variable so maintaining multiple values for the variable for different instances(different objects).

    Example:

```
    class dummy{
    int instanceVariableNumer; // valid and default valu will be 0.
    String instanceVariableString; // valid and default value will be null.
    AnotherClass anotherInstanceVariableObject; // valid and the value will be null by default;
    void someFunction(){
        int notInstanceVariable; // invalid as this is a local variable.
        int anotherNotInstanceVariable =10; // valid as the local variable has been initialized.
    }
}
```

7. Class variables are static variables and does not need to have an object to access. They also maintain only one state ie.. regardless of which instance(object) is calling the variable it will have same value across all the instances.

Inferring type using var:
var is not a reserved word, it is a reserved type name. So var can be used in declaring things like package name, method name, or variable name. It just cannot be used as a class name , interface name or an Enum name.

Example: this example compiles as the class name is Var and not var (case difference)

```
package var;
public class Var {
public void var() {
var var = "var";
}
public void Var() {
Var var = new Var();
}
}
```

1. can use var to declare local variable instead of type and the type will be automatically inferred.

    Example:

```
    void somFunction(){
    var number =7; // valid similar to int number = 7;.
    var someString = “Hello”; // valid similar to String someString = “Hello”;
    int number, var num =5; // invalid as more than one data type has been referenced in same
                                // line.
    var number = null; // invalid as the type null can simply be referred to a object but the
                            // compiler will not be sure of what is that object. So it is invalid.
    number = 5; // valid
    number = “Five”; // invalid as the type for number has been inferred as int;
}
```

2. Type Inference using var cannot be used in place of class or instance variables.

    Example:

```
    class SomeClass{
    var number = 8; //invalid as this is not a local variable.
   static var someString = “some string”; // invalid as it is not a local variable.
```

```
}
```

Example Scenario for var type inference:

```
        void varTypeInference(boolean check){
        var number;
        number = 10; //invalid for var the initialization should happen in the same line as the
                      //declaration. So var number = 10; is valid as the n is happening on same line.
        var number2; // here even though the number2 has been initialized on both success and
                      //failure scenarios of the if condition it is still invalid as the declaration is var.
                      //It will be valid if the declaration is int number2;
        if(check){
        number2 = 20;
        }
        else {
        number2 = 40;
        }
}

int varInMethodParameter( var number, var someNumber) // invalid as the number or
                                                      //  someNumber is not local variable.
{
return number + someNumber;
}
```

Variable Scope:

1. Local variables always have less than or equal to scope than the method they are defined in.

Example:

```
        public void eat(int piecesOfCheese){
                int bitesOfCheese = 1;
}
```

both piecesOfCheese and bitesOfCheese variables have scope only inside the method eat().

2. Instance variables will come to scope as soon as they have been defined and last for the entire life time of OBJECT itself.

3. CLASS variable aka STATIC Variables go into scope as soon as they have been declared and last for the entire lifetime of the program.

4. local variable : in scope from declaration till end of the block.
5. Method Parameters : in scope from declaration till the duration of the method
6. Instance variable : in scope from declaration till the object is eligible for garbage collection.
Class
7. Class variable: in scope from declaration till the program ends.

Destroying Objects:
Garbage collection refers to the process of automatically freeing memory on the heap by deleting objects that are no longer reachable in our program.

System.gc() method suggests Java JVM to run the Garbage collector but it does not guarantee that the garbage collector will be ran. Java JVM is free to ignore this command.

Garbage Collector Eligibility
An object will remain in heap until it is no longer reachable. ie..
i.) the object no longer has any references pointing to it.
ii.) all references to the object has gone out of scope.

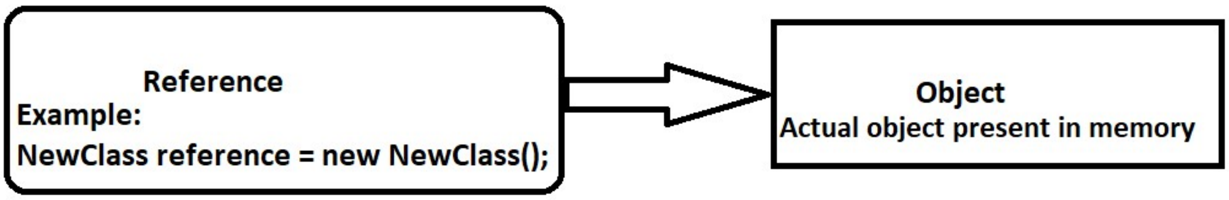
Reference
Example:
NewClass reference = new NewClass();
Object
Actual object present in memory

Fig 1. Object Reference image

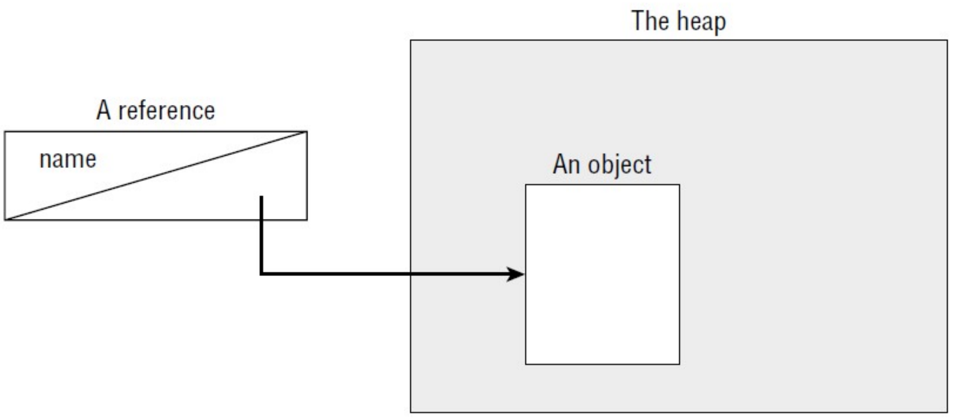

Fig 2 Object Reference mapping.

OBJECT vs Reference

It is the object that gets Garbage collected and not the Reference.
Constructors create Java objects

OBJECT	**Reference**
Sits on heap and does not have a name	Variable and has a name
No way to access except through a reference	Can be used to access contents of an object
Can not be assigned to another object, passed to a method or returned from a method.	Can be assigned to another reference, Passed to a method or returned from a method.
Consumes various amount of memory depending on the object	All references are of same size regardless of their types