Creating Java Classes:

1. A class in Java is just a blueprint telling what the objects created from it will look and act like.
2. Class is users define data types.
3. Class is a representation of similar kind of objects.
4. Class has these components:
    1. Instance variables.
    2. Static variables
    3. Instance methods.
    4. Static methods.
    5. Constructors. (For initialization and consistency)
    6. Inner classes
    7. Nested class
    8. Instance initialize Block
    9. Static initialize block
5. Class body is delineated by braces { }

Syntax:

```
<modifiers> class <name> <extends> <implements>
{       //constructors
        // member variables
        // member methods
        // nested class
        // block
}//end of class
```

Static:

1. These members are prefixed with static keyword.
2. These are associated with the class and class need not to be instanced to access these members.
3. These members are associated using either class name or object reference.
4. A static field is created when its class is loaded in memory.
5. A class is loaded when it is used first time during the execution of program.

Instance:

1. These members are associated with objects of the class and are accessible using object reference.
2. An instance field is created when an object of class is created.
3. Object is an imaginary boundary field of instance field.
4. Objects are created at runtime.
5. Class is created at compile time.

Different ways of creating object in java:

- **Using new keyword: -**

1. The new operator is used to create an object of class.
2. This is the most common way to create an object in java.

```
new <ClassName> (<parameters>)
```

3. The operator returns the reference of the object that we can store in the reference type variable.

```
Car c1= new Car ();
Car c2= new Car ();
```

**Note: c1 & c2 are reference variables of Car type (class is users define data type) not are objects. Object is created in heap memory.

- **Using Class.forName(): -**

1. If we know the name of the class & if it has a public default constructor, we can create an object in this way. It is also known as reflection.
2. Car car = (Car) Class.forName ("com.abc. Car").newInstance();

- **Using clone (): -**

1. The clone () can be used to create a copy of an existing object.

```
Car car1 = new Car ();
Car car = car1.clone ();
```

- **Using object deserialization: -**

1. Object deserialization is nothing but creating an object from its serialized form.

```
ObjectInputStream in = new OjectInputStream(anInputStream );
Car car = (Car) in.readObject();
```

- **<u>Using reflection</u>**:-(in another way)

  this.getClass ().getClassLoader ().loadClass ("com.abc.car").newInstance ();

<u>Accessing Member:</u>

There are three ways to accessing members of a class:

1. <ObjectReference>.<member> (both instance & static member)
2. <ClassName>.<member> (only static member)
3. <member> (from within class)

Suppose we have a Car class with 2 instance member variable, 1 instance member method, 1 static member variable and 1 static member method.

```
public class Car {

    int width;
    int height;
    static float washCharge;

    void move() {
        // TODO: Write your own code
    }

    static void avg() {
        // TODO: Write your own code
    }
}
```

```
Car c1=new Car();
Car c2=new Car();

c1.height=10;
c2.width=15;

Car.washCharge=50;

Car.height=20;

c1.washCharge=100;
```

Compile time error

Warning

TOKENS:

Lexical tokens are the basic building blocks of a source code.

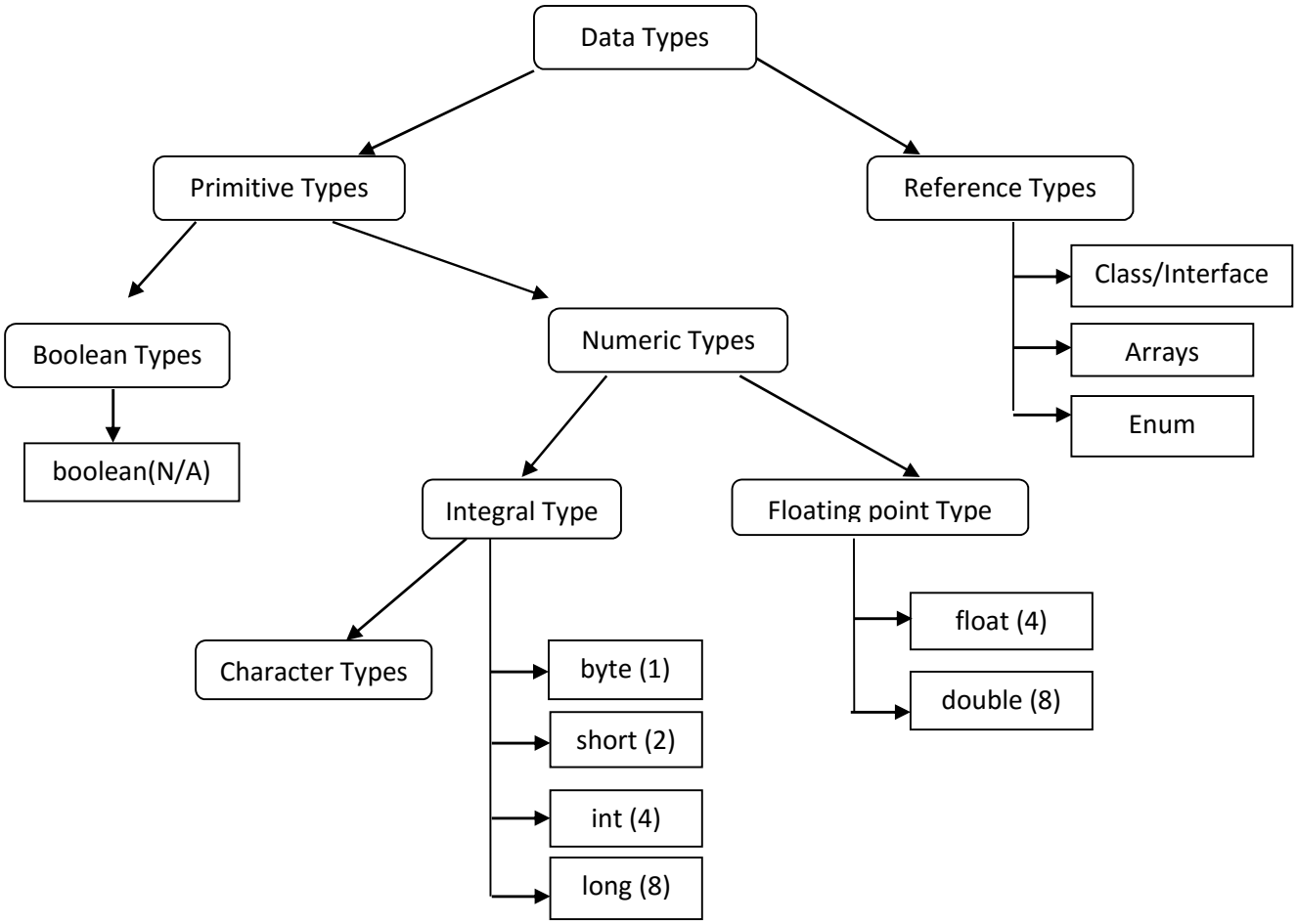
- Keywords (are defined in the compiler)
- Identifiers (these are the name of variable, method, class etc. they must not begin with a digits and contain digits, letters underscore, currency symbols.)
- Literals (values)
  - Integer (0b10, 10, 010, 0x10, 10L)

    *default data type is int
  - Floating points (3.0,-0.03, 10e-5, 20E+5)

    *default data type is double
  - Boolean (true, false)
  - Character ('a','1','@')
  - Escape Sequence
  - String ("abc","123")
- White Space (Tab,enter,space bar)
- Operators
- Comments
  - //Single line comment
  - /* Multi line comment*/
  - /** Java documentation comment*/

ESCAPE SEQUENCE:

\n	→ new line
\r	→ carriage return
\t	→ tab
\b	→back slash
\f	→ form feed
\'	→
\"	→
\\	→



Member variable vs. Local variable:

1. Member variable are declared in class body. Local variables are declared in any block.
2. Local variables are not initialized by default.
3. A member variable is initialized by their default value.
4. It is an error to used uninitialized variable.

Default values:

TYPES	VALUES
Boolean	false
Charactor	'\u0000'
Integer	0
Floating point	0.0D/0.0F
Reference	null

Java Coding CONVENSION (not a rule):

1. **Name of class/interface:** My, MyFirst, MyFirstClass
2. **Name of variable/method:** my, myFirst, myFirstVar
3. **Name of constant/enum:** MY, MY_FIRST, MYFIRST_CONST

The main() method:

```
public static void main (String [] args) {

        // All code goes here...
}
```

1. Called by JVM which is external entity, so it is public.
2. There is no need to make it object, hence it is static.
3. It is accessible by class name.

Printing in Java:

System.*out*.print(*data*); → data

System.*out*.println(*data*); → data + /n

- System is a class
- out is a static reference field in System class
- print() or println() are methods

1. When we pass a literals / primitive values in System.*out*.print(*data*), it is converted to String and print it.
2. When we pass a reference values then toString() method of the Object is called to convert the reference value into String.

Java source code structure:

i. <Package declaration>
ii. <import declaration>
iii. <class and interfaces declaration>

1. All these are optional.
2. The name of the source code must be name of the public class/interface declared in it. Otherwise it can be anything.
3. To compile the source code, we can use javac command. the syntax is:

```
javac <args> <Source fileName >
```

4. The compiler checks the source code for error and generates a **.class** file for every class / interfaces declared in it.
5. These **.class** file contain the byte code of the corresponding class / interfaces.
6. To execute a java application, we can use java command. the syntax is:

```
java <args> <Class Name having main> <args>
```

7. The command creates the JVM and passes everything that is given to it.
8. The JVM loads the specified class in memory and called its main method.