3. Java Language Structure

Data types

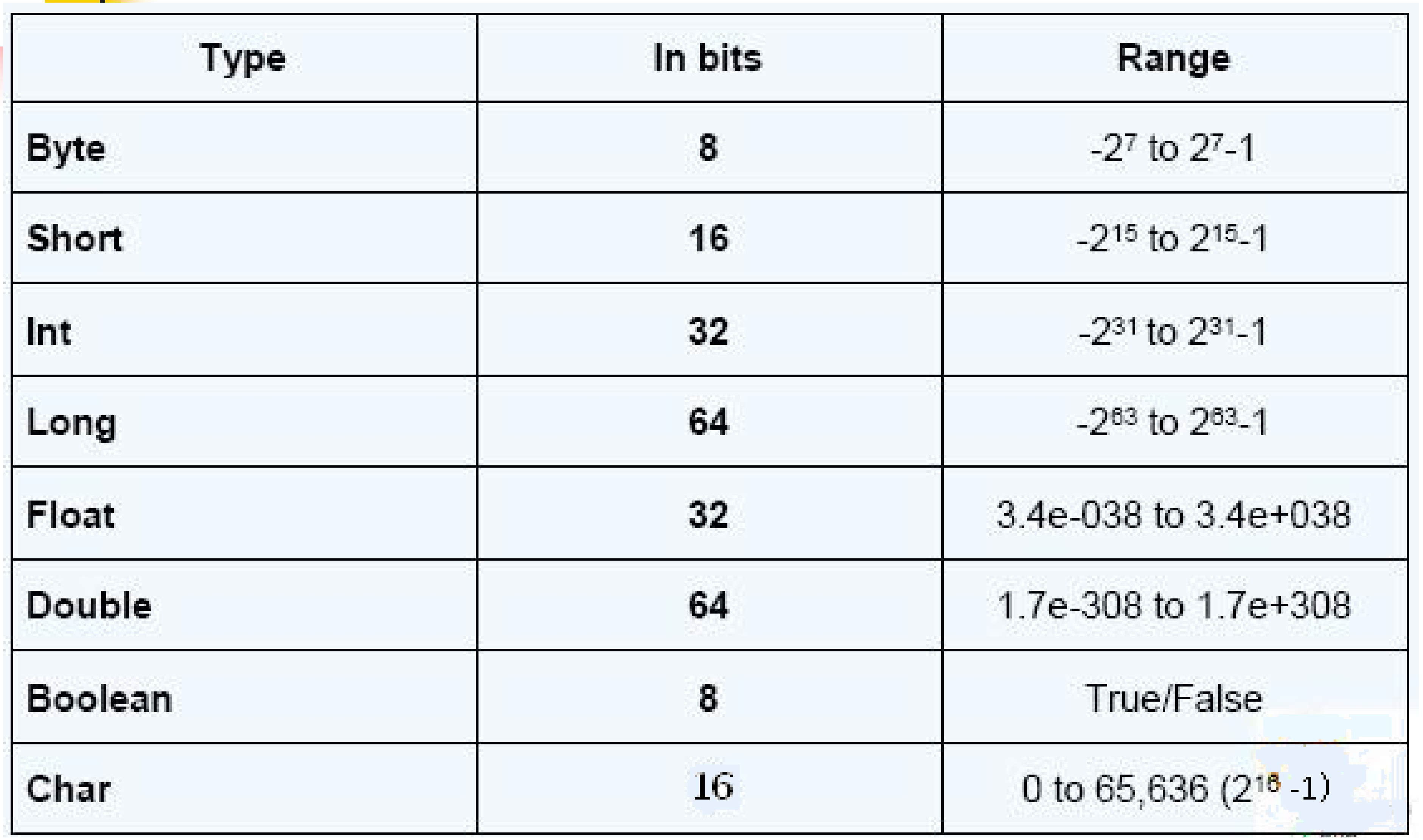
Operators

Control Statements

Data Type in Java

- Java is a Strongly typed language
- Two types:
  - Primitive type
  - Reference type
- Reference types cannot be cast to primitive types

Java defines eight primitive (or simple) types of data



Type	In bits	Range
Byte	**8**	-2^{7} to $2^{7}-1$
Short	**16**	-2^{15} to $2^{15}-1$
Int	**32**	-2^{31} to $2^{31}-1$
Long	**64**	-2^{63} to $2^{63}-1$
Float	**32**	3.4e-038 to 3.4e+038
Double	**64**	1.7e-308 to 1.7e+308
Boolean	**8**	True/False
Char	16	0 to 65,636 (2^{16} -1)

Java Data Types

- Since the java programs finally run on the JVM, the size of the data types remain same irrespective of the platform on which the programs are executed.
- Primitive types default to a certain value when declared within a class but must be explicitly initialized within methods.

Integers

- There are 4 ways of representing integer data.
  - byte
  - short
  - int
  - long
- Stores whole numbers.
- All of these are signed, Java does not support unsigned integers.

Floating-point Types

- Floating-point numbers are used for numbers with a decimal part.
- There are two floating-point types:
  - float (32 bits, single-precision)
  - double (64 bits, double-precision).

Characters

- Stores a character data.
- Represented as - char
- Characters in Java are Unicode (16 bits)

Boolean

- The boolean type can have one of two values: ***true*** or ***false***.
- Note that unlike in other **C**- like languages, boolean is not a number, nor can it be treated as one.
- All tests of boolean variables should test for *true* or *false*.

Variables

- A variable is a named memory location that can hold various values.
- A variable needs to be declared before it could be used.
- Declaration consists of:

```
<variable type> <variable name> [= initial value];
```

Example:

```
boolean  myFlag = true;
```

Variables

- The type may be primitive type or an Object type.
- We can declare multiple variables of the same type in the same declaration statement.

  Example:

  ```
  int  a = 54,  b,  c;
  ```

  Declares 3 variables of integer type.

Variables

- The Variable name may:
  - consists of letters, digits, **$** or **_**
  - not start with digits.

$$\left.\begin{array}{l}\text{my\$Money}\\ \text{\$andRupee}\\ \text{\_get\_35\_counter}\end{array}\right\}\text{Valid identifiers}$$

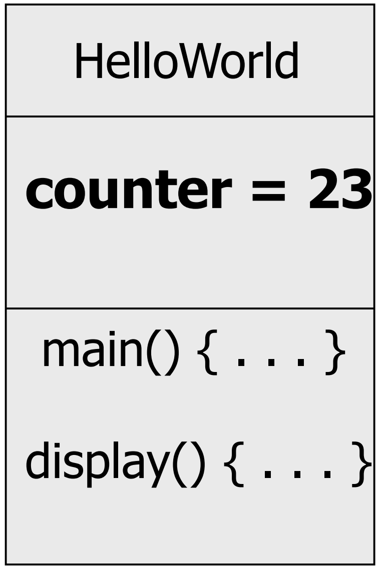
$$\text{35Counters}\xrightarrow{\quad}\text{Invalid Identifier}$$

```
public class TypeTester {
   public static void main(String args[])  {
        char c;           /* Declaration of char variable*/
        int i;            /* Declaration of int variable*/
        float f;          /* Declaration of float variable*/
        double d;         /* Declaration of double variable*/
        c='A';
        i=10;
        f=12.2400f;
        d=24.4888848009;
        System.out.println("\n The Value stored in c is:"+ c);
        System.out.println("\n The Value stored in i is:"+ i);
        System.out.println("\n The Value stored in f is:"+ f);
        System.out.println("\n The Value stored in d is:"+ d);
   }
}
```

Scope and Lifetime of Variables

- Scope of a variable depends on where we declare it.
  - Instance Variable
  - Local Variable
- We can even limit the scope into a block of statements enclosed by braces.

Scope and Lifetime of Variables

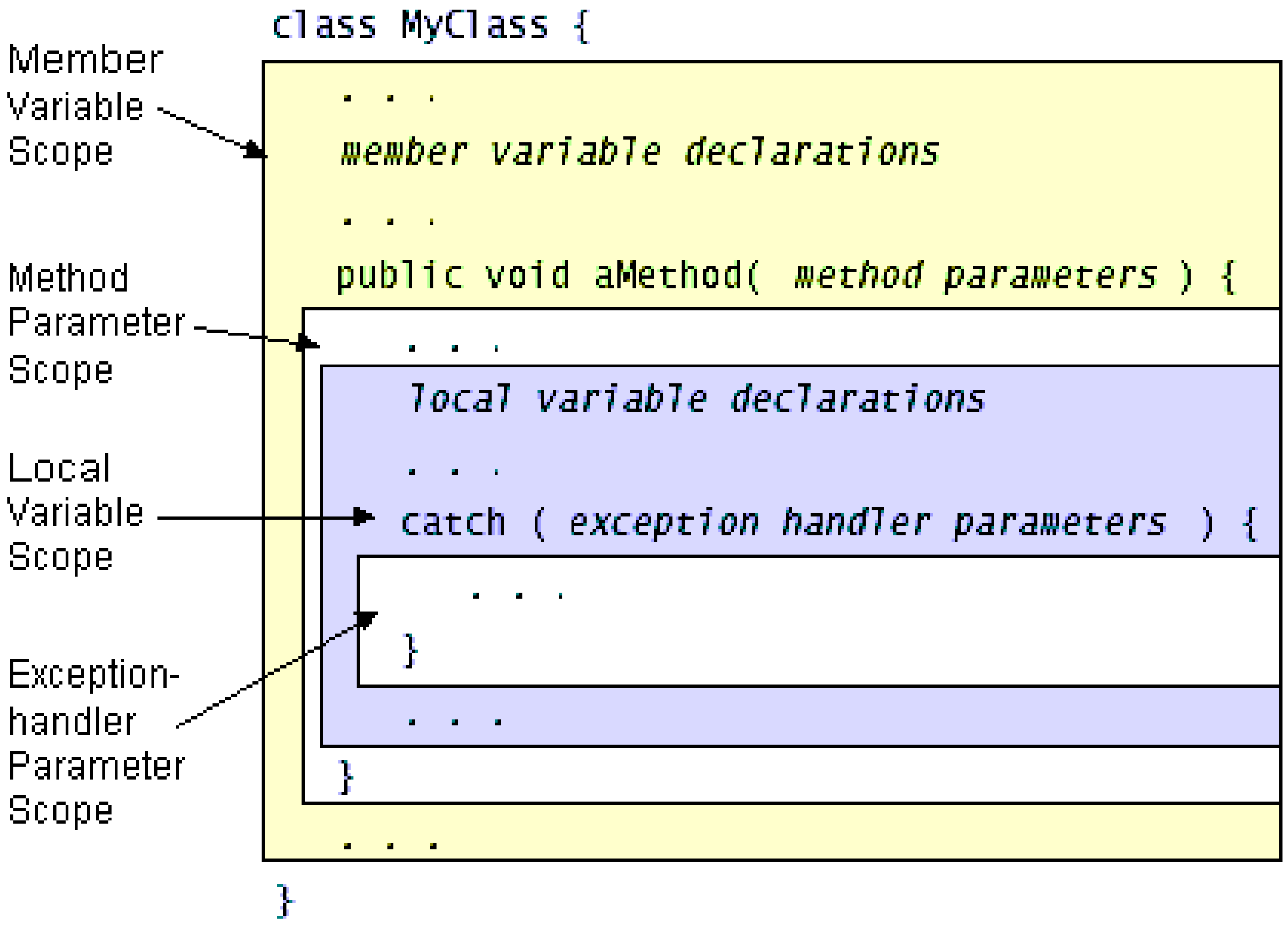


```
public class HelloWorld  {
        int counter = 23;    // instance variable

        public static void main (String[] args)  {
             int number = 42;    // local variable
             System.out.println("Counter:"+counter);
             System.out.println("Number:"+number);
        }

        public void display() {
             System.out.println("Counter:"+counter);
          // System.out.println("Number:"+number);
}
```

Scope and Lifetime of Variables



```
class MyClass {
    . . .
    member variable declarations
    . . .
    public void aMethod( method parameters ) {
        . . .
        local variable declarations
        . . .
        catch ( exception handler parameters ) {
            . . .
        }
        . . .
    }
    . . .
}
```

Scope and Lifetime of Variables

- An instance variable is the one which is created when an instance or an object is created, and is accessible from any method in the class.
- A local variable is alive as long as the control is within the method / block.
- An instance variable is automatically initialized to its default value
- A local variable must initialized explicitly before it is used.

Type Conversion and Casting

- Implicit Conversion (Automatic type Conversion)
- Explicit Conversion (Type Casting)

Implicit Conversion

- Take place if the following 2 conditions are met:
  - The 2 types are compatible
  - The destination type is larger than the source type
- This is also known as '*widening conversion*'.
- The numeric types are not compatible with char or boolean.

$$\text{byte} \longrightarrow \text{short} \longrightarrow \text{int} \longrightarrow \text{long} \longrightarrow \text{float} \longrightarrow \text{double}$$

Example:

```
short index = 35;
long number;
number = index;
```

Implicit Conversion

- Java defines several type promotion rules that applies to expressions.
- If an expression has operands belonging to different types, the entire expression evaluates to a widest type.

Example:

```
double result = (f * b) + (i / c) – (d * s);
```

Explicit Conversion (Type Casting)

- No automatic type conversion when the source type is larger than the destination type.
- To create a conversion between two incompatible types we must use a cast.

  (target-type) value;

- This conversion is called a *narrowing conversion.*

Example:

```
int  a;
byte  b;
b = (byte) a;
```