Variables and Standard Data Types

Java Developer

StayAhead Training; April, 2022

Variable Review*

Variables and Standard Data Types

- A variable is a meaningful name for some data stored in memory
- Some variables store values, some store references to objects
- Variable declaration, e.g. `String myName;`
- Variable initialisation, e.g. `myName = "Stuart";`
- Variable declaration + initialisation, e.g. `var myName = "Stuart";`
- Note that var may only be used where the data type of the variable can be inferred from the value assigned (var cannot be used for fields)

Fields vs Local Variables

Variables and Standard Data Types

- A field is the specification of an item of state, and…
- A field is an attribute of an object, and…
- A fields is a variable that is declared inside a class but outside of any method
- A local variable is one that is declared inside a method inside a class
- Local variables are visible/accessible only within the method
- Parameters are local variables too
- Classes can, and often do, have fields and local variables that share names

Primitive Variables

Variables and Standard Data Types

- A primitive variable is one that stores a value
- There are eight primitive types only
- Each primitive data type is a keyword, e.g. int
- All primitive types are numeric though boolean and char variables are represented as true/false and ascii/unicode characters respectively
- Primitive variables are passed by (copy of) value

Reference Variables

Variables and Standard Data Types

- A reference variable is one that stores a reference to an object
- A reference is effectively the memory address of an object
- Every reference data type is a class, e.g. String
- Each class you create is a new reference data type
- Reference variables are passed by (copy of) reference

Primitive Types

Variables and Standard Data Types

Type	Size	Min Value	Max Value
boolean	1 bit	false	true
byte	1 byte	-128	127
short	2 bytes	-32,768	32,767
char	2 bytes	0	65535
int	4 bytes	-2,147,483,648	2,147,483,647
long	8 bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
float	4 bytes	1.4E-45	3.4028235E+38
double	8 bytes	5E-324	+∞

Literals

Variables and Standard Data Types

- A literal is value, e.g.

```
var myBoolean = true;    // true is a boolean literal

var myInt = 1;           // 1 is an integer literal

var myString = "Hello"; // "Hello" is a String literal
```

- NB: Strings are an exceptional case in Java; they are objects but a String variable can be initialised like a primitive variable

Default Types

Variables and Standard Data Types

- Each variable has a data type but so too do literal values
- The default integral type is **int**

```
var x = 1;  // x is an int

byte b = x; // compilation error (4 bytes won't go into 1)
```

- The default floating point type is **double**

```
var y = 0.5; // y is a double

float f = y; // compilation error (8 bytes won't go into 4)
```

Suffixes*

Variables and Standard Data Types

- You can specify the type of some literals by adding a suffix
- l/L = long
- f/F = float
- d/D = double

```
var y = 0.5f; // y is a float

float f = y;  // OK
```

Underscores*

Variables and Standard Data Types

- The underscore may be used in a number literal to make it easier to read, e.g.

```
var bigNum = 1_000_000;
```

- The following usages are illegal:

```
var illegal1 = 1._5; // adjacent to decimal point

var illegal2 = 5_000_000_000_L; // adjacent to suffix

var illegal3 = _42; // at the beginning

var illegal4 = 42_; // at the end
```

Casting

Variables and Standard Data Types

- Casting is the process of creating a variable of type y from a variable of type x
- Casting applies to both primitive and reference variables
- BUT the casting of reference variables pertains to inheritance etc.

```
var myInt = 97;

var myChar = (char) myInt; // myChar = 'a'
```

- Take care not to corrupt your data, e.g. by casting int to byte/short

Reference Types

Variables and Standard Data Types

- Every class is a reference type, so there are thousands of them
- The most common reference types are array and String
- Each new class you create is a new reference type

Null Pointer*

Variables and Standard Data Types

- null is a Java keyword that means references nothing
- A reference variable that is declared but not initialised is null by default
- A reference type field is likewise null by default
- Any attempt to access a member using a variable assigned to null will result in a runtime error (your app will crash), e.g.

```
String myName;

System.out.println(myName.toLowerCase()); // runtime error
```

Arrays

Variables and Standard Data Types

- An array is an object/a collection of values/references of the same type
- Arrays are indexed (ordered) with the first element at index 0
- Arrays are fixed-size; they cannot grow nor shrink
- Array elements may be overwritten

```
var names = new String[3]; // size = 3

names[0] = "Tom";

var firstName = names[0];
```

Strings

Variables and Standard Data Types

- A String is an object/an array of characters
- A String object can be created in one of two ways, e.g.

```
var myFirstString = new String("Hello");

var mySecondString = "Hello"; // preferred
```

- String objects are immutable - they cannot be changed
- The String class has lots of methods for processing String data

Primitive Wrappers

Variables and Standard Data Types

- Each primitive type has an associated wrapper class
- E.g. the int type's wrapper class is named Integer
- A wrapper class instance wraps a value, e.g.

```
var wrappedInt = new Integer(1);
```

- Java collections, e.g. ArrayList, cannot store values, only references
- Wrapper classes help to overcome that limitation

Autoboxing and Autounboxing*

Variables and Standard Data Types

- The Java interpreter will automatically convert between value and wrapper object and vice-versa
- From value to wrapper object is autoboxing, e.g.

```
Integer wrappedInt = 1;
```

- From wrapper object to value is autounboxing, e.g.

```
int unwrappedInt = wrappedInt;
```

Constants

Variables and Standard Data Types

- A constant is a variable that cannot be reassigned
- The final keyword is used to make a field/local variable constant
- A final variable must be initialised at declaration time
- NB: a constant reference variable does not mean the referenced object cannot be changed

```
final var meaningOfLife = 42;

meaningOfLife = 43; // compilation error
```

Type Inference

Variables and Standard Data Types

- Java is statically typed - a variable's data type cannot change
- Pre Java 10, variable declaration required explicit type specification, e.g.

```
String myName = "Stuart";
```

- Since Java 10 the var keyword can be used where the data type can be inferred from the value assigned

```
var myName = "Stuart";
```

- NB: var cannot be used in the specification of fields, even where the field is assigned an initial value

Varargs

Variables and Standard Data Types

- Varargs = variable number of arguments
- Sometimes, when coding a method, you want the caller to be able to pass in a variable number of arguments, e.g.

```
double getAverage(int... nums) {
  // instructions
}
```

- The ellipsis (three dots) indicates that the caller may pass in zero or more int arguments which will be wrapped inside an array (nums is an `int[]`)