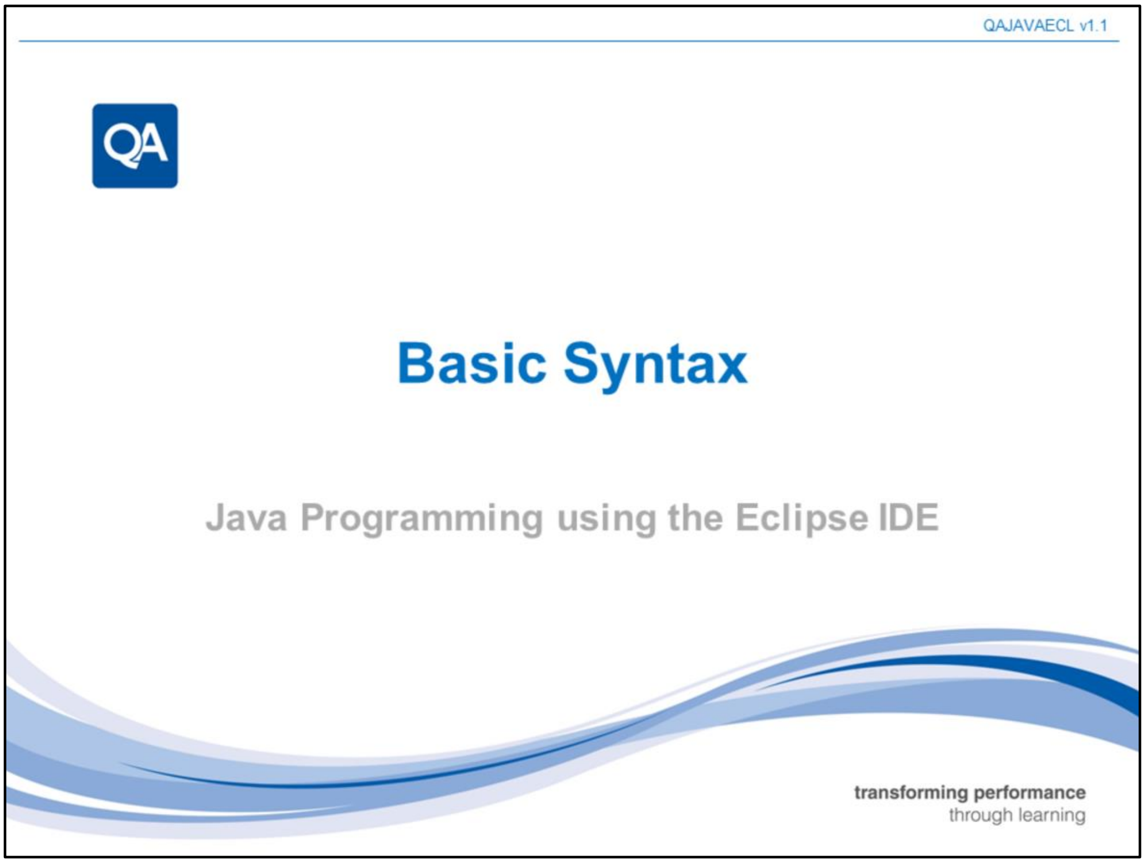
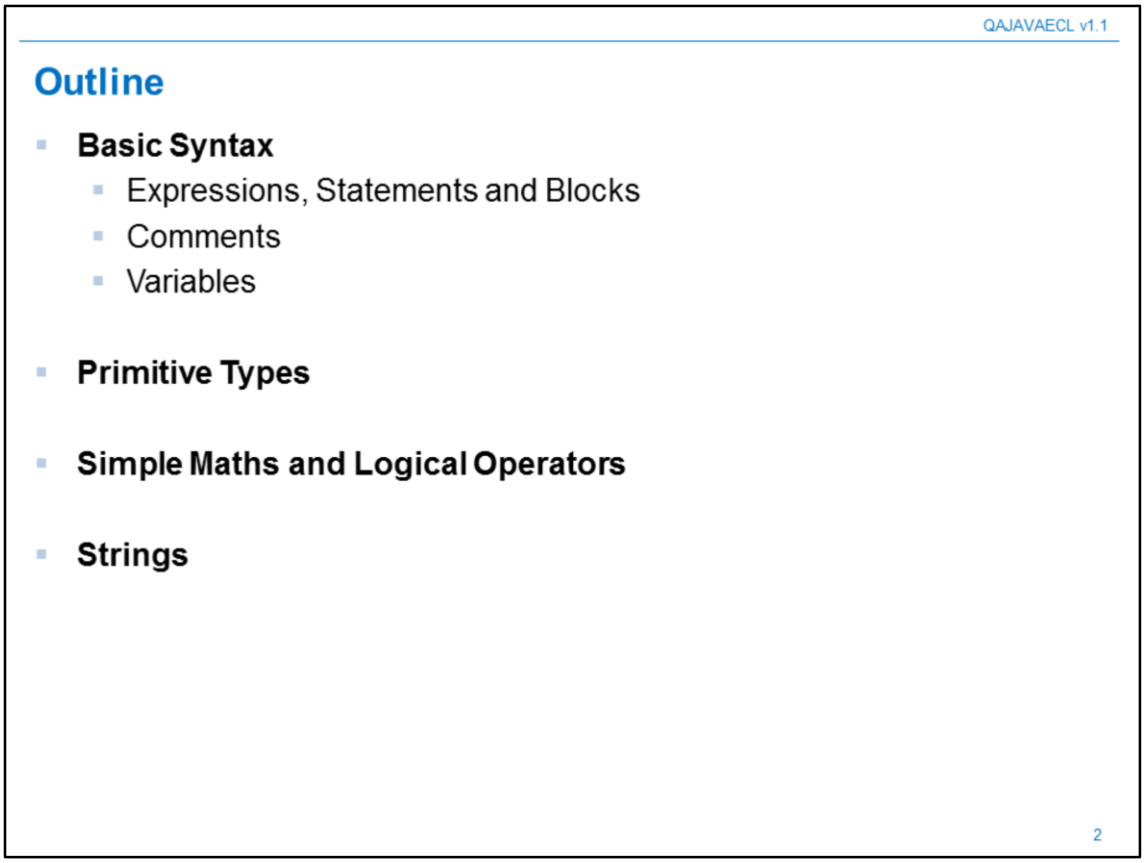


QAJAVAECL v1.1

# Basic Syntax

Java Programming using the Eclipse IDE

transforming performance
through learning

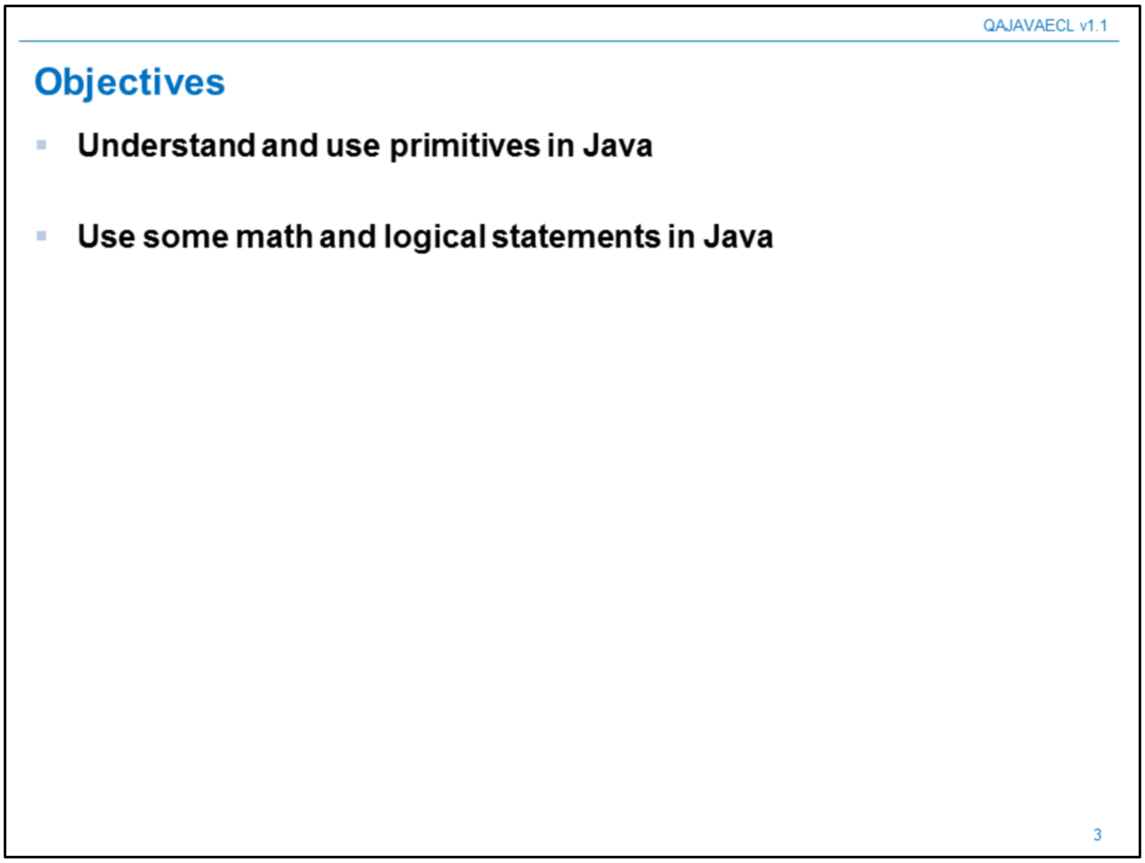


QAJAVAECL v1.1

## Outline

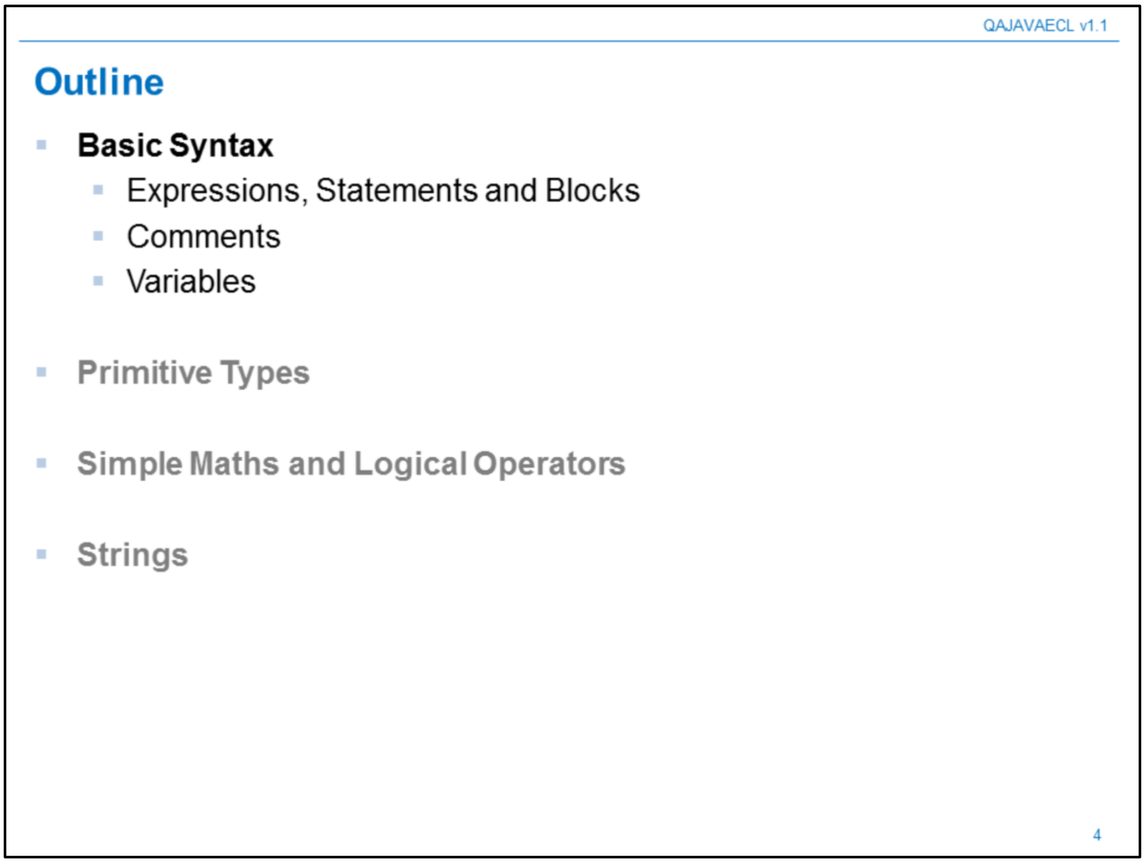
- **Basic Syntax**
  - Expressions, Statements and Blocks
  - Comments
  - Variables
- **Primitive Types**
- **Simple Maths and Logical Operators**
- **Strings**

2



## Objectives

- Understand and use primitives in Java
- Use some math and logical statements in Java

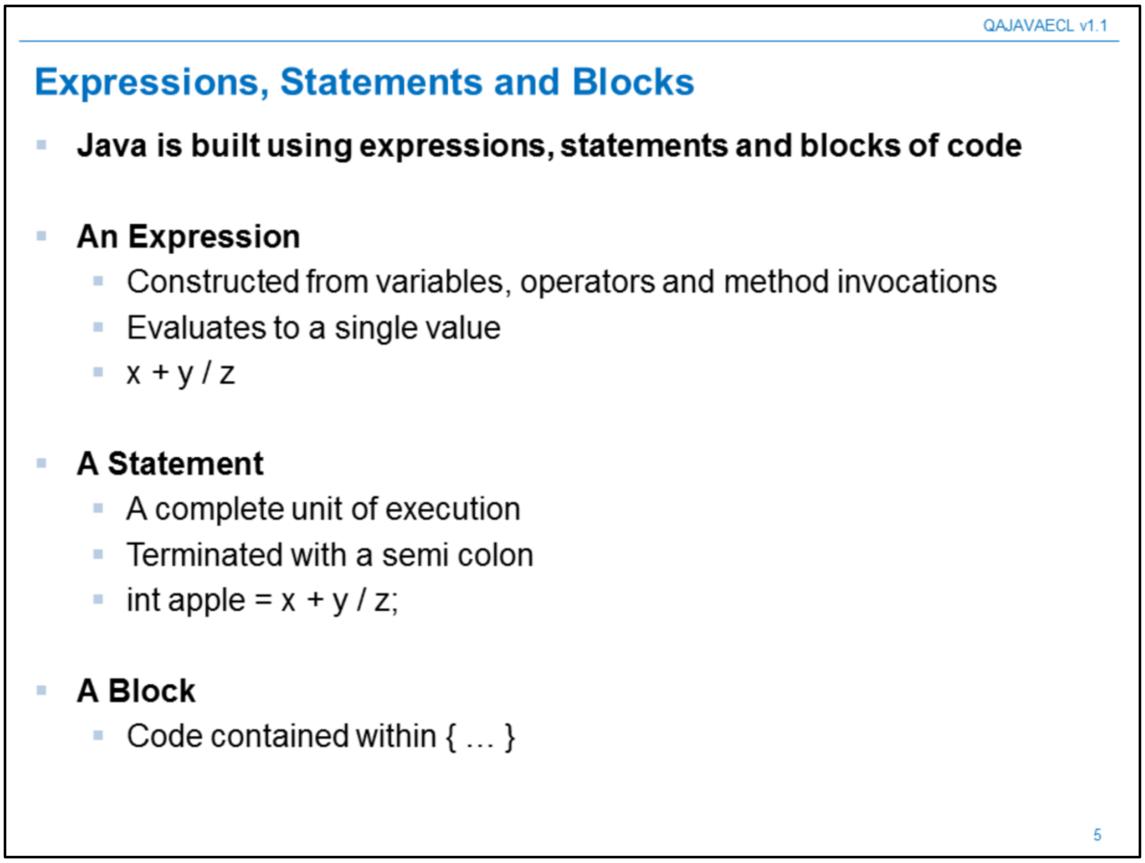


QAJAVAECL v1.1

## Outline

- **Basic Syntax**
  - Expressions, Statements and Blocks
  - Comments
  - Variables
- **Primitive Types**
- **Simple Maths and Logical Operators**
- **Strings**

4

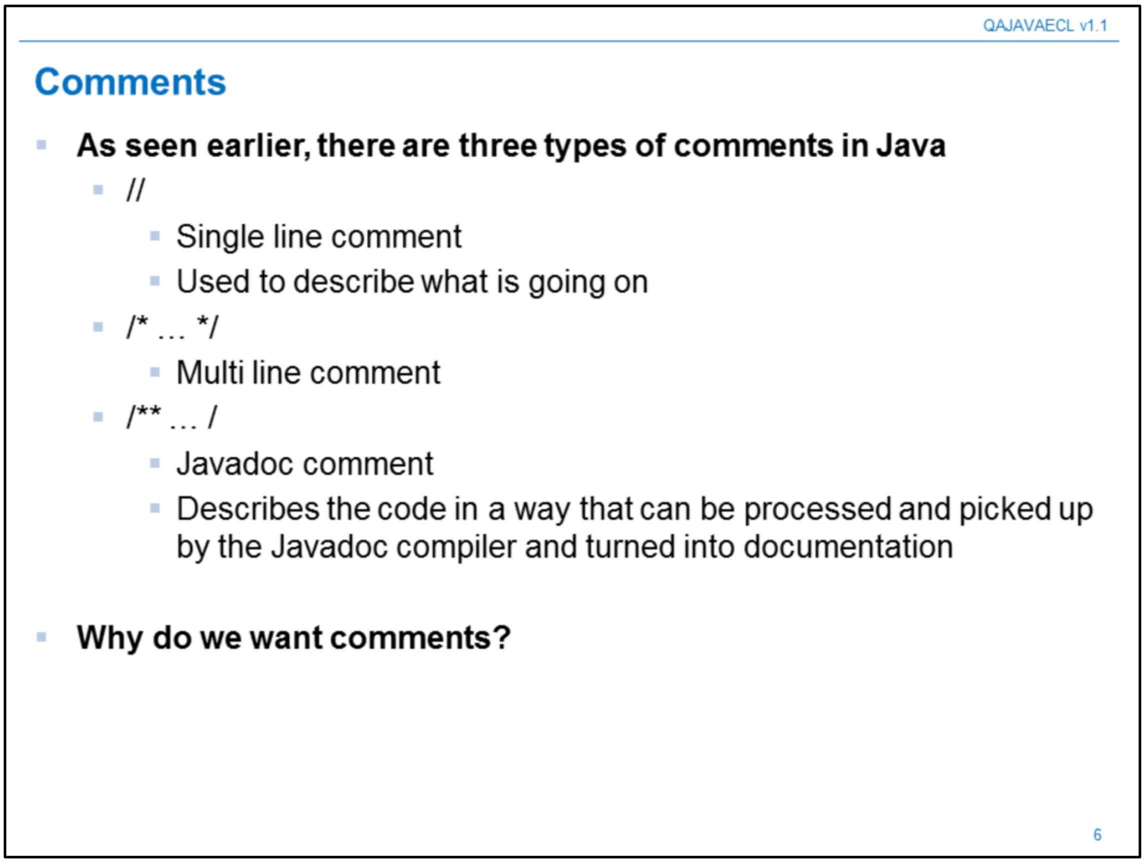


QAJAVAECL v1.1

## Expressions, Statements and Blocks

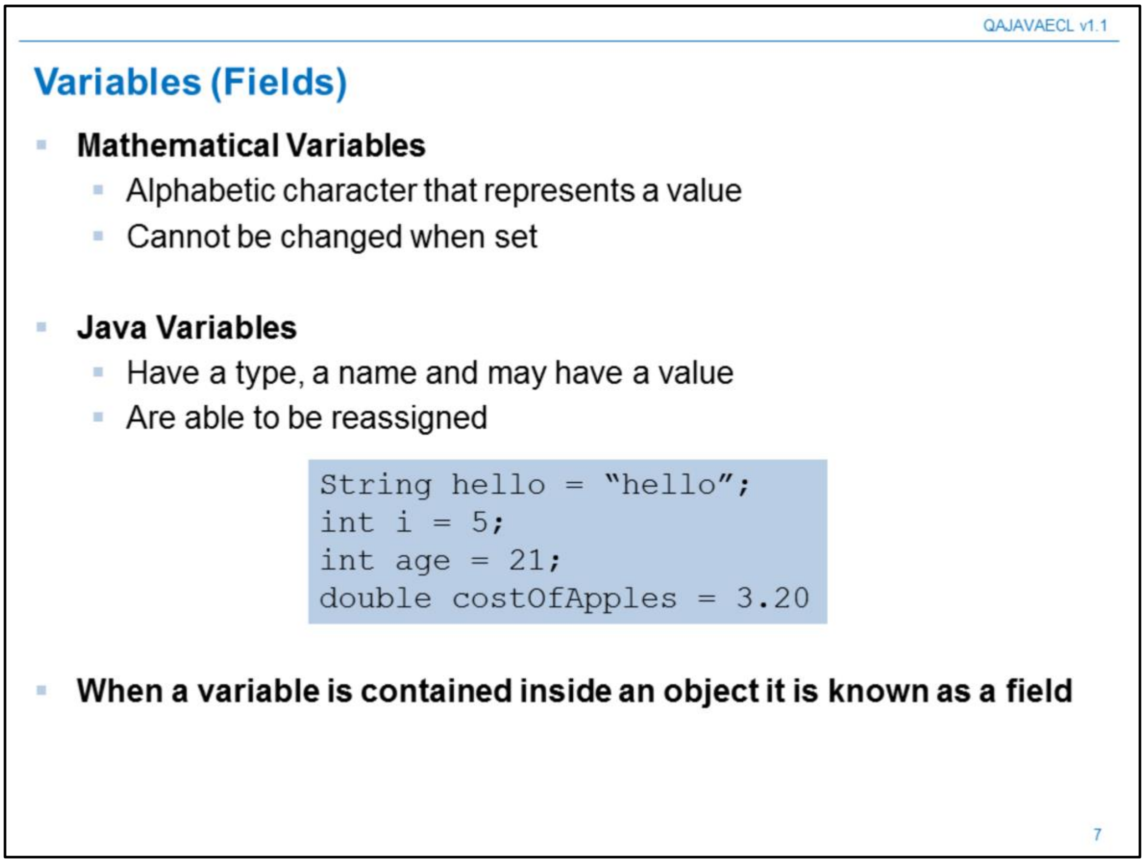
- **Java is built using expressions, statements and blocks of code**
- **An Expression**
  - Constructed from variables, operators and method invocations
  - Evaluates to a single value
  - x + y / z
- **A Statement**
  - A complete unit of execution
  - Terminated with a semi colon
  - int apple = x + y / z;
- **A Block**
  - Code contained within { … }

5



## Comments

- **As seen earlier, there are three types of comments in Java**
  - //
    - Single line comment
    - Used to describe what is going on
  - /* … */
    - Multi line comment
  - /** … /
    - Javadoc comment
    - Describes the code in a way that can be processed and picked up by the Javadoc compiler and turned into documentation
- **Why do we want comments?**



QAJAVAECL v1.1

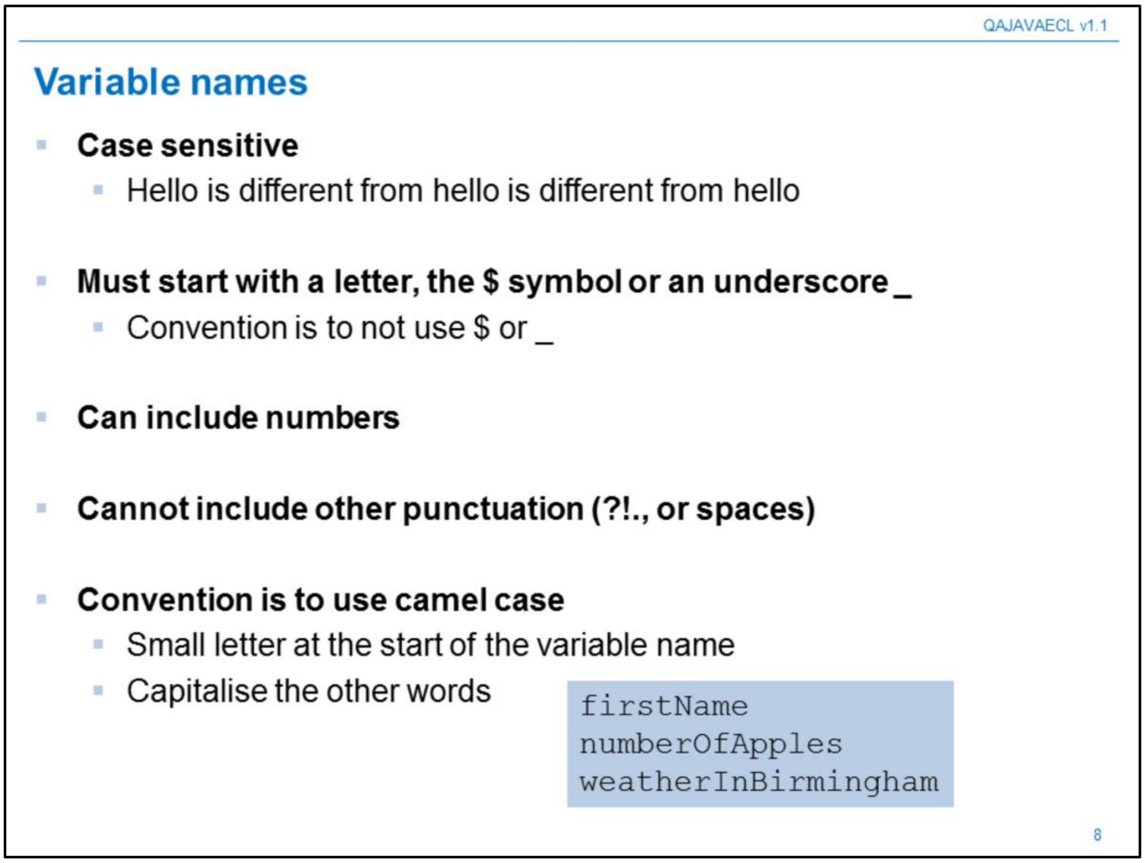
## Variables (Fields)

- **Mathematical Variables**
  - Alphabetic character that represents a value
  - Cannot be changed when set
- **Java Variables**
  - Have a type, a name and may have a value
  - Are able to be reassigned

```
String hello = "hello";
int i = 5;
int age = 21;
double costOfApples = 3.20
```

- **When a variable is contained inside an object it is known as a field**

7



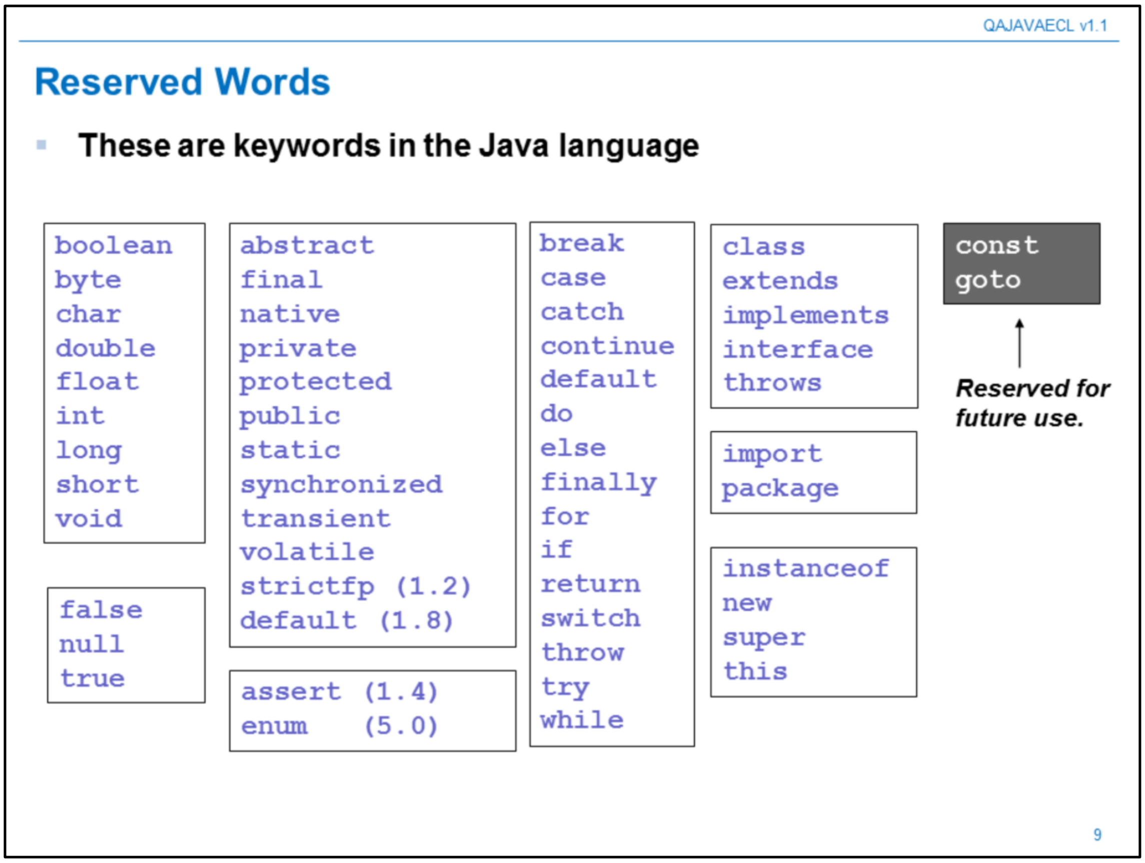
QAJAVAECL v1.1

## Variable names

- **Case sensitive**
  - Hello is different from hello is different from hello
- **Must start with a letter, the $ symbol or an underscore _**
  - Convention is to not use $ or _
- **Can include numbers**
- **Cannot include other punctuation (?!., or spaces)**
- **Convention is to use camel case**
  - Small letter at the start of the variable name
  - Capitalise the other words

```
firstName
numberOfApples
weatherInBirmingham
```

8



QAJAVAECL v1.1

## Reserved Words

- These are keywords in the Java language

```
boolean     abstract           break      class        const
byte        final              case       extends      goto
char        native             catch      implements
double      private            continue   interface
float       protected          default    throws
int         public             do
long        static             else       import
short       synchronized       finally    package
void        transient          for
            volatile           if         instanceof
false       strictfp  (1.2)    return     new
null        default  (1.8)     switch     super
true                           throw      this
            assert  (1.4)      try
            enum    (5.0)      while
```

***Reserved for future use.***

9

All the words on this slide are reserved words in the Java language and cannot be used as variable names. Most of the words on the slide count as Java keywords, but you should be aware that true, false and null are regarded as predefined values rather than keywords. This esoteric point is only mentioned because in the Sun Certified Programmer exam, a question sometimes crops up which asks you to identify the keywords from a list which includes the word null - and you are not supposed to pick null as the answer!

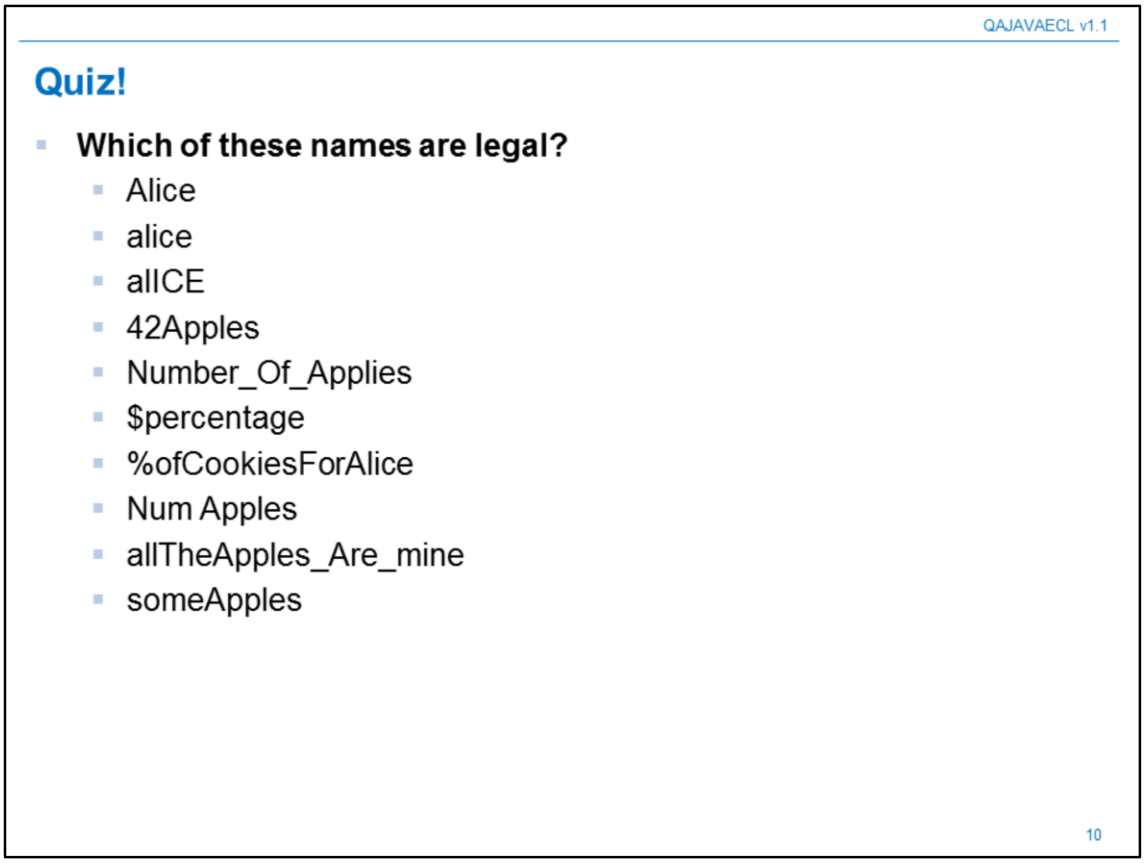
Previously the list of words "reserved for future use" included

byvalue, cast, future, generic, inner, operator, outer,rest, and var

and most development environments still highlight these even though they will still compile as identifiers within your code.

strictfp was introduced in Java 1.2. See the language specification from Sun for a detailed description of what it does.

Assert was introduced in Java 1.4 and requires a compiler flag to enable assertion checking.

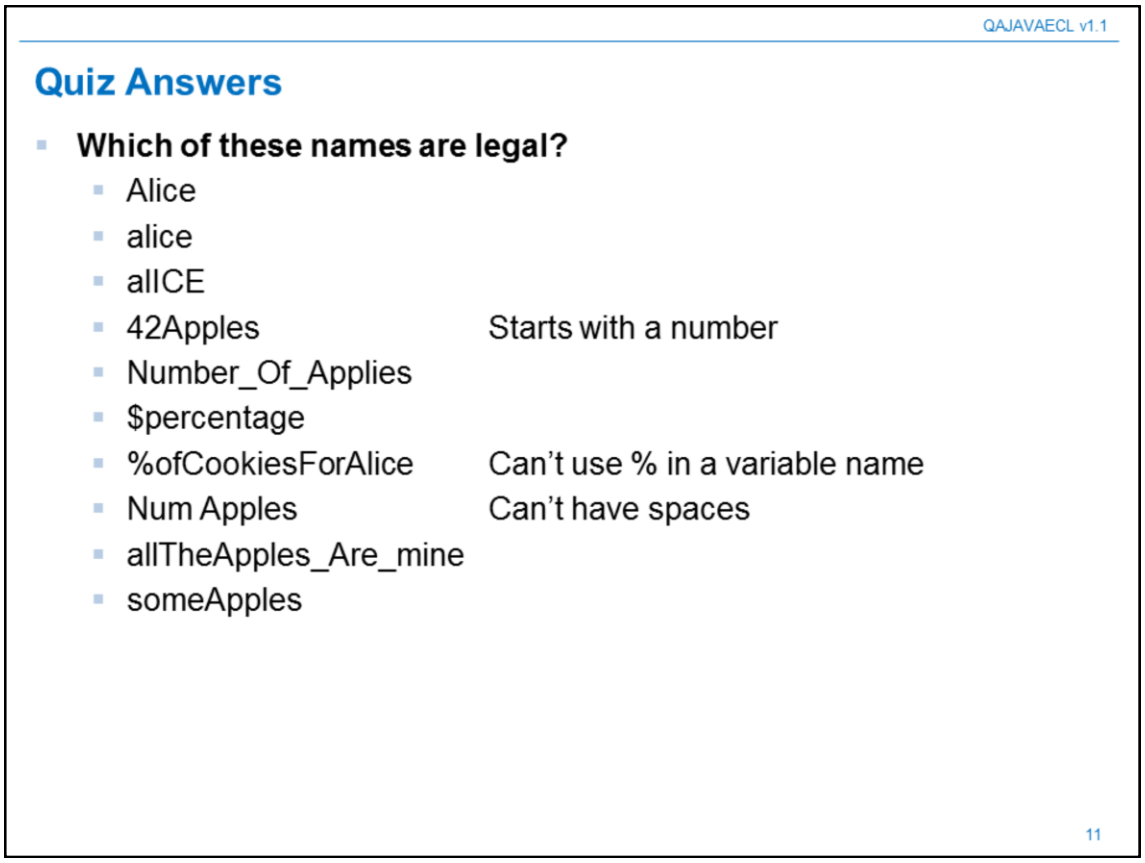


QAJAVAECL v1.1

## Quiz!

- **Which of these names are legal?**
  - Alice
  - alice
  - alICE
  - 42Apples
  - Number_Of_Applies
  - $percentage
  - %ofCookiesForAlice
  - Num Apples
  - allTheApples_Are_mine
  - someApples

10



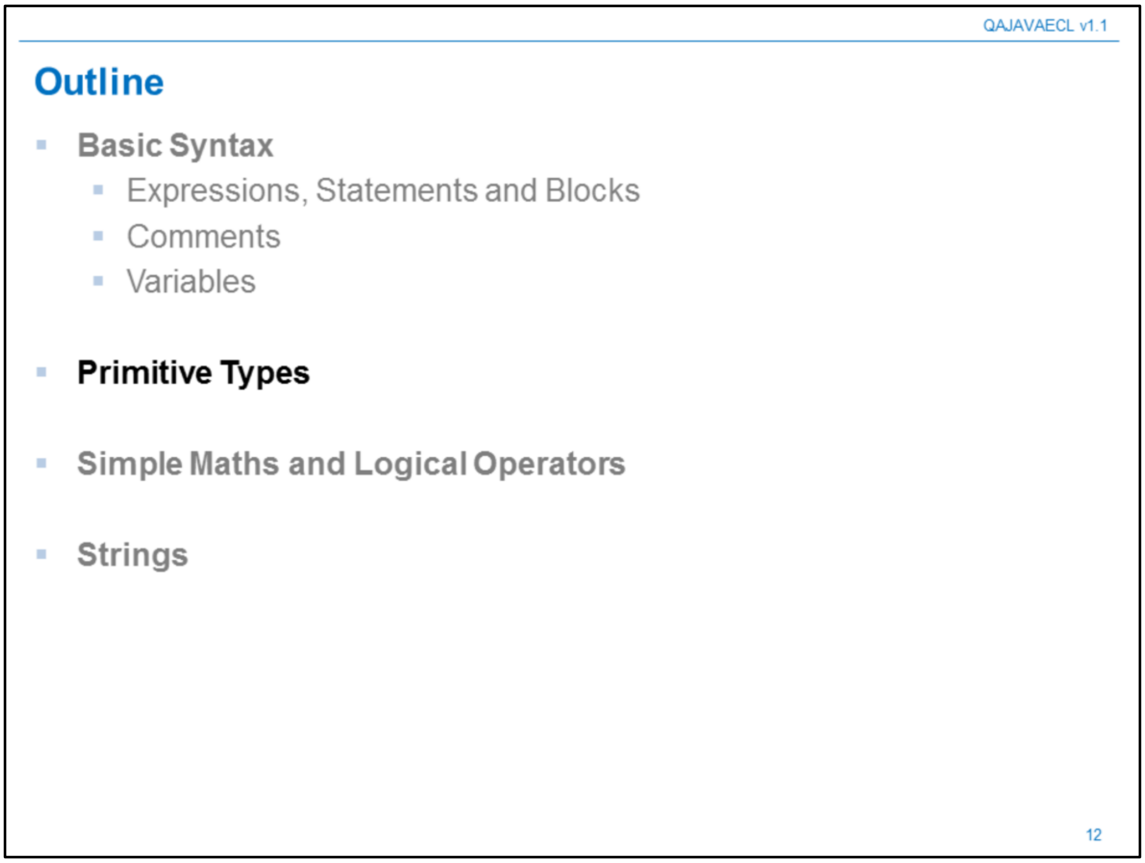
QAJAVAECL v1.1

## Quiz Answers

- **Which of these names are legal?**

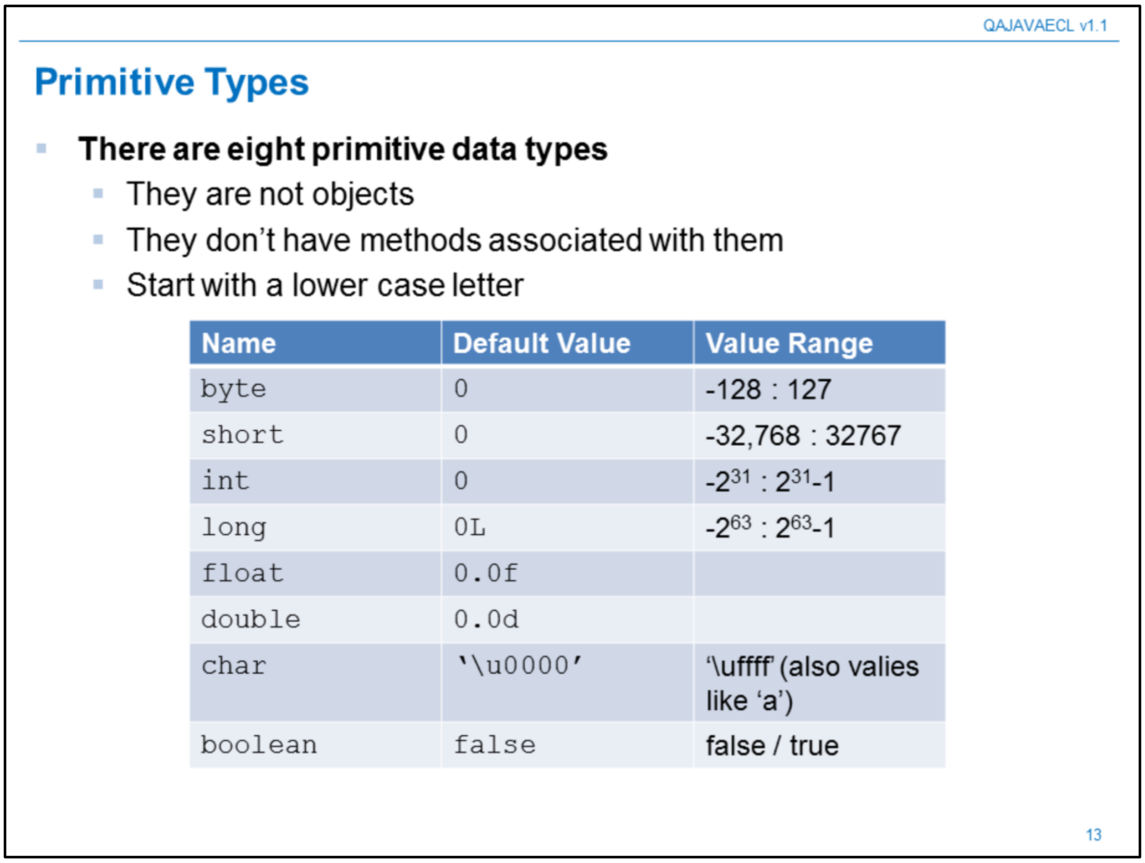
| | |
|---|---|
| Alice | |
| alice | |
| allCE | |
| 42Apples | Starts with a number |
| Number_Of_Applies | |
| $percentage | |
| %ofCookiesForAlice | Can’t use % in a variable name |
| Num Apples | Can’t have spaces |
| allTheApples_Are_mine | |
| someApples | |

11



## Outline

- **Basic Syntax**
  - Expressions, Statements and Blocks
  - Comments
  - Variables
- **Primitive Types**
- **Simple Maths and Logical Operators**
- **Strings**



QAJAVAECL v1.1

## Primitive Types

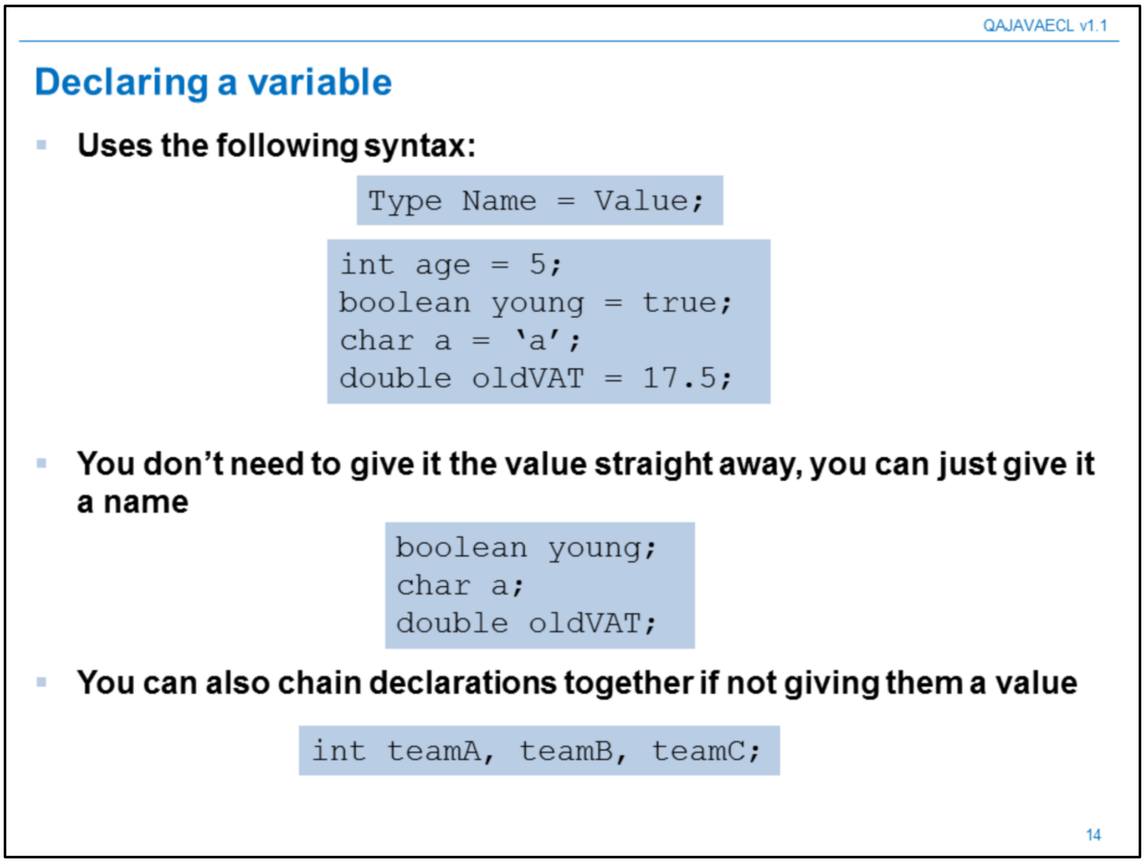
- **There are eight primitive data types**
  - They are not objects
  - They don't have methods associated with them
  - Start with a lower case letter

| Name | Default Value | Value Range |
|---|---|---|
| `byte` | `0` | -128 : 127 |
| `short` | `0` | -32,768 : 32767 |
| `int` | `0` | $-2^{31} : 2^{31}-1$ |
| `long` | `0L` | $-2^{63} : 2^{63}-1$ |
| `float` | `0.0f` | |
| `double` | `0.0d` | |
| `char` | `'\u0000'` | '\uffff' (also valies like 'a') |
| `boolean` | `false` | false / true |

13

It's not as easy to give ranges for floats and doubles as it depends on the specification used by the JVM. In general a float uses 32bit single precision and a double uses 64 bit single precision.

See http://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3 for the full specification in java floats and doubles. The level of precision can be important for some applications (finance especially) but for this course we don't need to worry about it!



## Declaring a variable

- Uses the following syntax:

```
Type Name = Value;
```

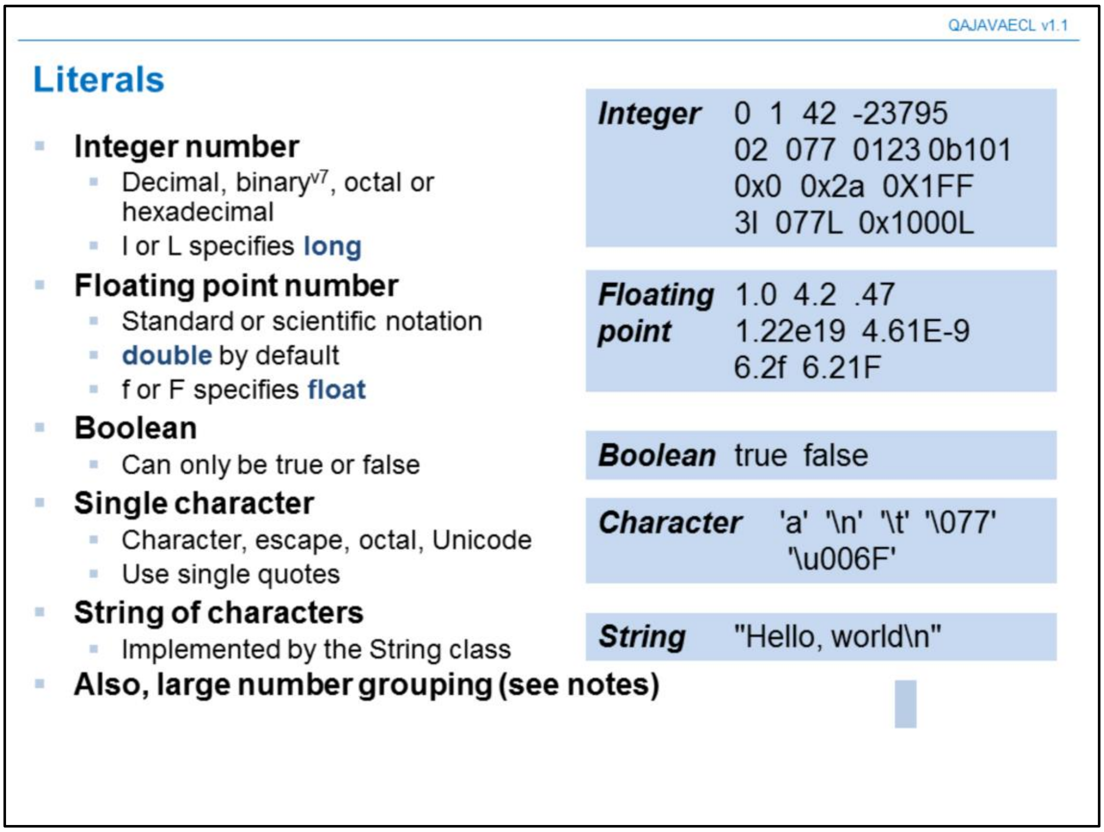
```
int age = 5;
boolean young = true;
char a = ‘a’;
double oldVAT = 17.5;
```

- You don’t need to give it the value straight away, you can just give it a name

```
boolean young;
char a;
double oldVAT;
```

- You can also chain declarations together if not giving them a value

```
int teamA, teamB, teamC;
```



QAJAVAECL v1.1

## Literals

- **Integer number**
  - Decimal, binary$^{v7}$, octal or hexadecimal
  - l or L specifies **long**
- **Floating point number**
  - Standard or scientific notation
  - **double** by default
  - f or F specifies **float**
- **Boolean**
  - Can only be true or false
- **Single character**
  - Character, escape, octal, Unicode
  - Use single quotes
- **String of characters**
  - Implemented by the String class
- **Also, large number grouping (see notes)**

| | |
|---|---|
| ***Integer*** | 0 1 42 -23795<br>02 077 0123 0b101<br>0x0 0x2a 0X1FF<br>3l 077L 0x1000L |
| ***Floating point*** | 1.0 4.2 .47<br>1.22e19 4.61E-9<br>6.2f 6.21F |
| ***Boolean*** | true false |
| ***Character*** | 'a' '\n' '\t' '\077'<br>'\u006F' |
| ***String*** | "Hello, world\n" |

You can use *literals* to specify simple values. It is important to know how to specify literals in various formats for all the primitive types and for strings.

By default, integer literals are 32-bit signed numbers. Literals with leading zeros are octal (base 8), literals with a leading 0x or 0X are hexadecimal (base 16) and finally, literals with a preceding 0b are binary. The last facility was added in JSE 7. The characters A through F in a hexadecimal number can be in either upper or lower-case. Integer literals greater than 0x7FFFFFFF are taken as `long`. You can force any integer literal to be `long` by appending an L to it (upper or lower-case).

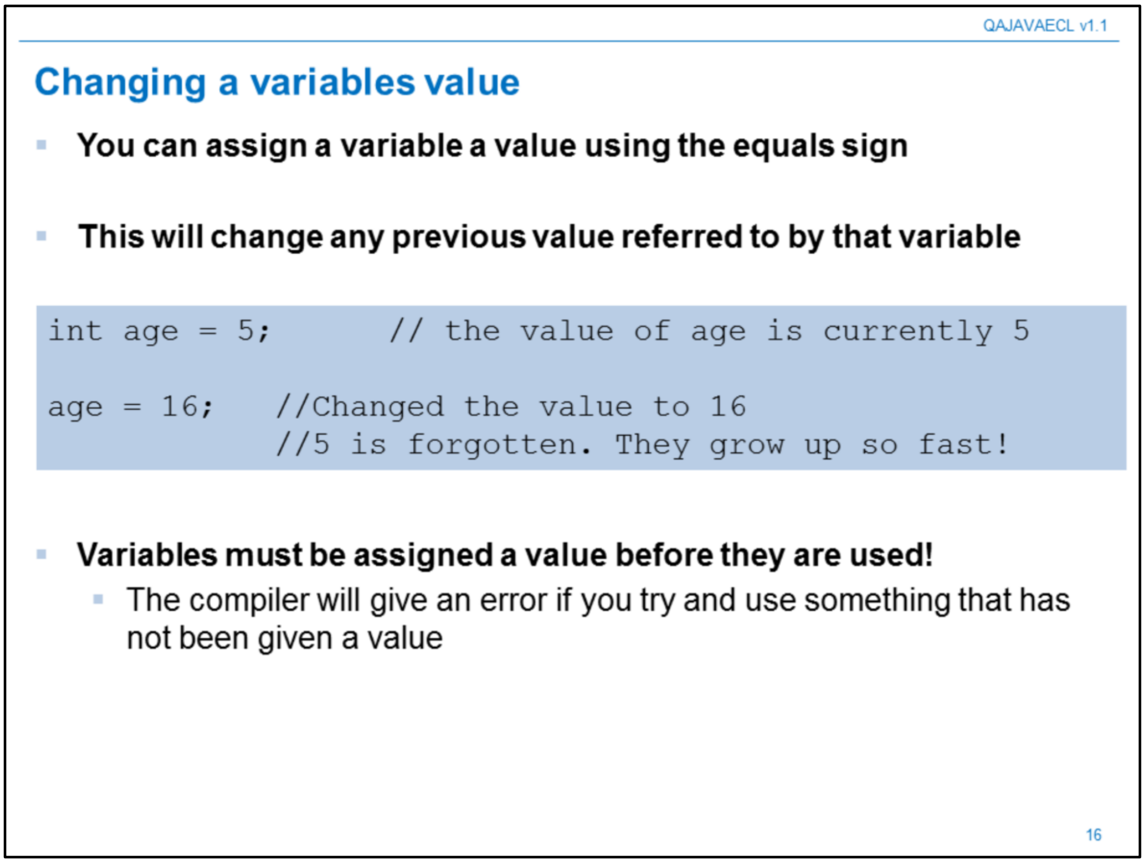
Floating-point literals can be specified in either standard (32.1) or scientific (3.21e2) notation. By default, floating-point literals are taken as double precision. You can force any floating-point literal to single precision by appending an F to it (upper or lower-case).

Boolean literals can be either `true` or `false`. Both `true` and `false` are keywords in the language, so you do not need to define them elsewhere. Note that `true` and `false` are not numeric values and cannot be converted to integers or vice versa.

Character literals are normally printable characters enclosed in *single* quotes, e.g. 'a'. To specify a non-printable character such as a new line or a special character such as a single quote, you must specify either its octal or hexadecimal value, or use its corresponding *escape sequence*. An escape sequence consists of the backslash character followed by another character; some examples are '\n' for new line, '\t' for a single tab, '\'' for a single quote and '\\' for a backslash. As mentioned previously, `char` types are stored as 16 bit Unicode characters. To specify a character literal in its Unicode representation, prefix its 4-digit code with \u, e.g. '\u0027'

String literals consist of any number of characters inside *double* quotes. Strings can also contain escape codes such as \n . In Java, strings are implemented by a class called `String`, but this is largely transparent. When you use a string literal like "Hello, world", Java automatically creates a `String` object for you (i.e., you do not need to use the `new` operator).

JSE7 introduced a means of separating groups of three digits with the underscore to help the readability of large integers. For example the integer 10010101 may be written as 10_010_101



## Changing a variables value

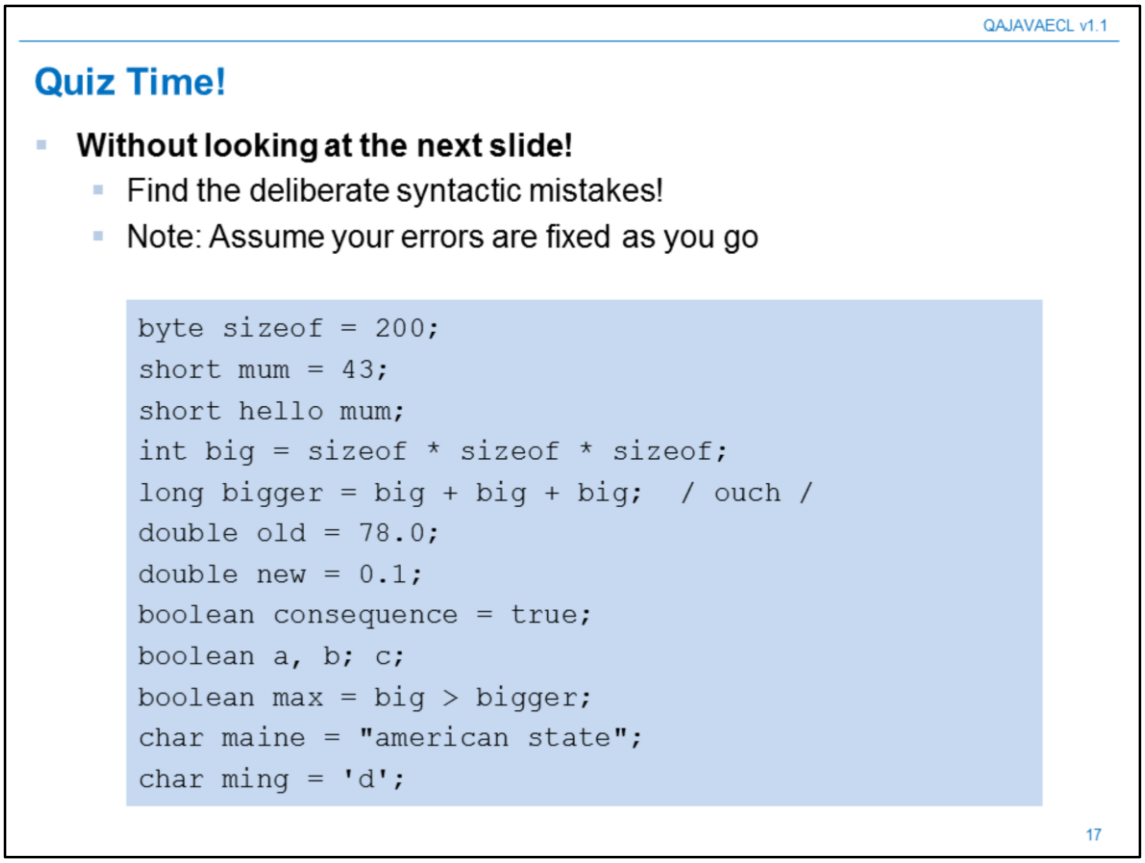
- **You can assign a variable a value using the equals sign**
- **This will change any previous value referred to by that variable**

```
int age = 5;       // the value of age is currently 5

age = 16;    //Changed the value to 16
             //5 is forgotten. They grow up so fast!
```

- **Variables must be assigned a value before they are used!**
  - The compiler will give an error if you try and use something that has not been given a value

16



QAJAVAECL v1.1

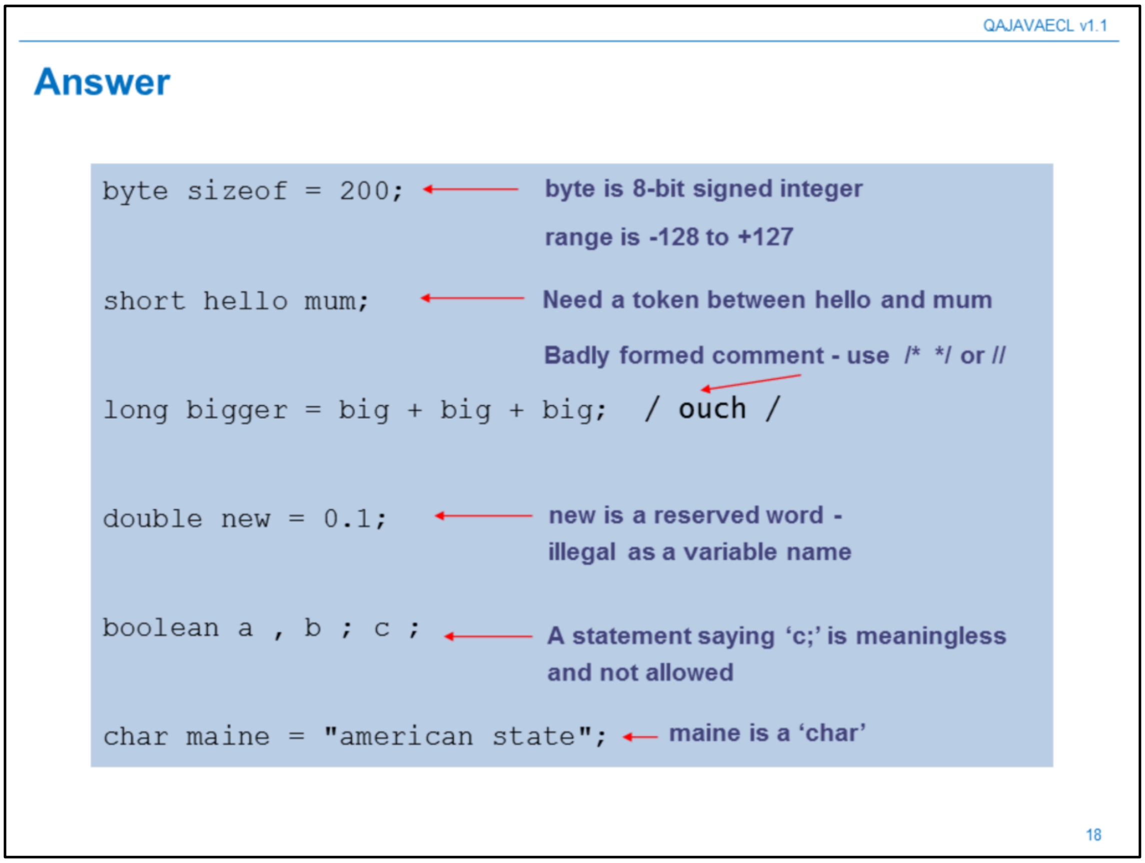
## Quiz Time!

- **Without looking at the next slide!**
  - Find the deliberate syntactic mistakes!
  - Note: Assume your errors are fixed as you go

```
byte sizeof = 200;
short mum = 43;
short hello mum;
int big = sizeof * sizeof * sizeof;
long bigger = big + big + big;  / ouch /
double old = 78.0;
double new = 0.1;
boolean consequence = true;
boolean a, b; c;
boolean max = big > bigger;
char maine = "american state";
char ming = 'd';
```

17

Hopefully, you'll never write code like this!



## Answer

```
byte sizeof = 200;
short hello mum;
long bigger = big + big + big;   / ouch /
double new = 0.1;
boolean a , b ; c ;
char maine = "american state";
```

byte is 8-bit signed integer
range is -128 to +127
Need a token between hello and mum
Badly formed comment - use /* */ or //
new is a reserved word - illegal as a variable name
A statement saying 'c;' is meaningless and not allowed
maine is a 'char'

The variable sizeof is of type byte, and can therefore only hold numbers in the range -128 to +127. There are no unsigned integer built-in types in Java. If you're familiar with C or C++, note that sizeof is not a reserved word in Java.

The line short hello mum needs a token between hello and mum. There are many ways to fix this error depending on what you want to do. You could separate them with an assignment operator or a comma.

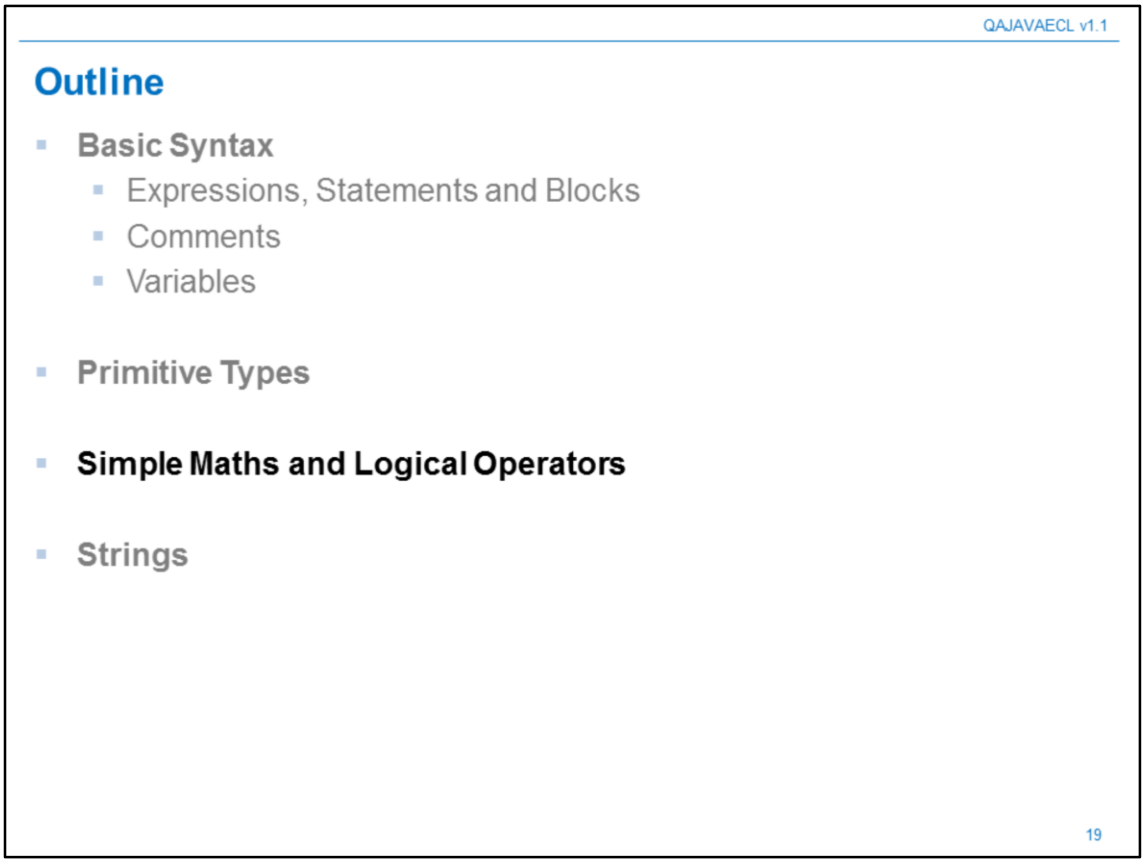
In the bigger statement, the add operations are fine, i.e. adding big to big, to big. However, the comments are wrongly defined. You must use either // to end of line, or /* . . . */.

In Java, new is a reserved word.

The boolean line separates b from c with a semicolon rather than a comma. The compiler would pick this up as an error even if c has been declared, as a variable name by itself is an invalid statement.

Characters cannot  hold more than one character. To declare a string, you must create an object of the string class, as follows:

**String maine = "american state";**

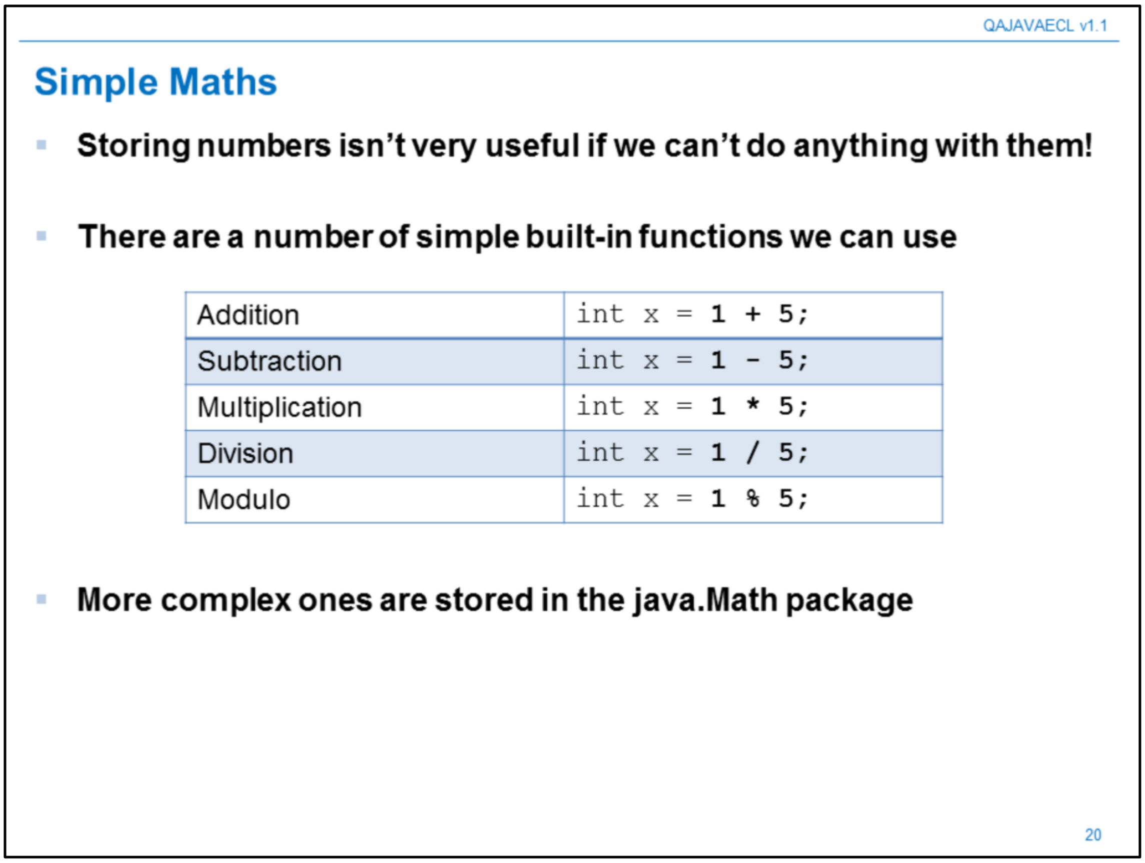


QAJAVAECL v1.1

## Outline

- **Basic Syntax**
  - Expressions, Statements and Blocks
  - Comments
  - Variables
- **Primitive Types**
- **Simple Maths and Logical Operators**
- **Strings**

19



QAJAVAECL v1.1

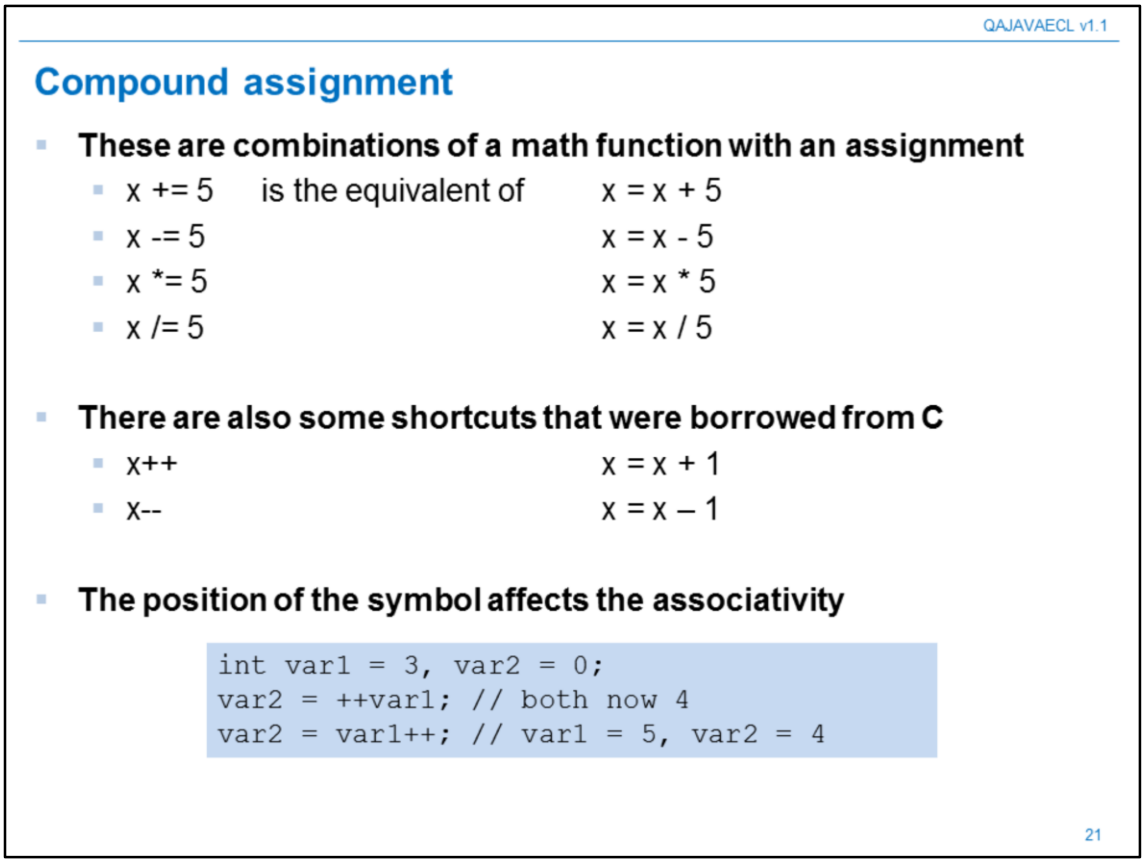
## Simple Maths

- Storing numbers isn't very useful if we can't do anything with them!
- There are a number of simple built-in functions we can use

| | |
|---|---|
| Addition | `int x = 1 + 5;` |
| Subtraction | `int x = 1 - 5;` |
| Multiplication | `int x = 1 * 5;` |
| Division | `int x = 1 / 5;` |
| Modulo | `int x = 1 % 5;` |

- More complex ones are stored in the java.Math package

20



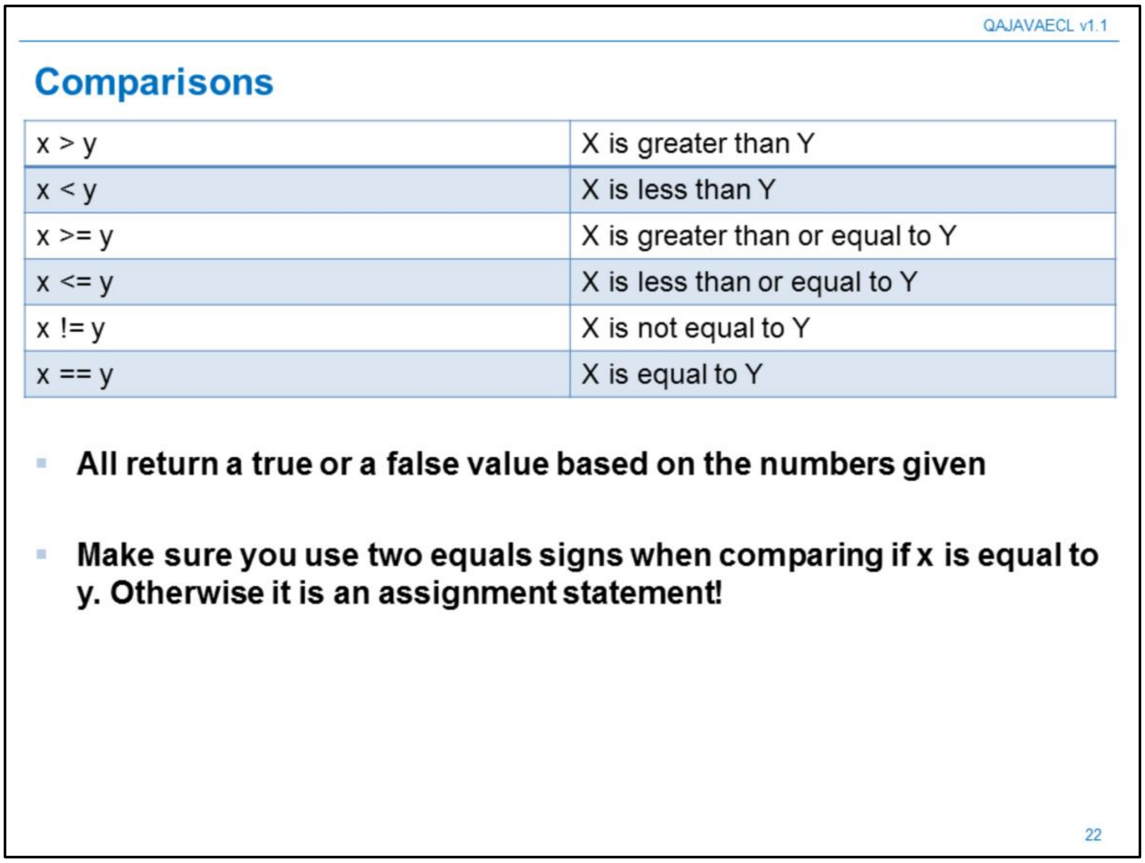
QAJAVAECL v1.1

## Compound assignment

- **These are combinations of a math function with an assignment**
  - x += 5 is the equivalent of x = x + 5
  - x -= 5 x = x - 5
  - x *= 5 x = x * 5
  - x /= 5 x = x / 5
- **There are also some shortcuts that were borrowed from C**
  - x++ x = x + 1
  - x-- x = x – 1
- **The position of the symbol affects the associativity**

```
int var1 = 3, var2 = 0;
var2 = ++var1; // both now 4
var2 = var1++; // var1 = 5, var2 = 4
```

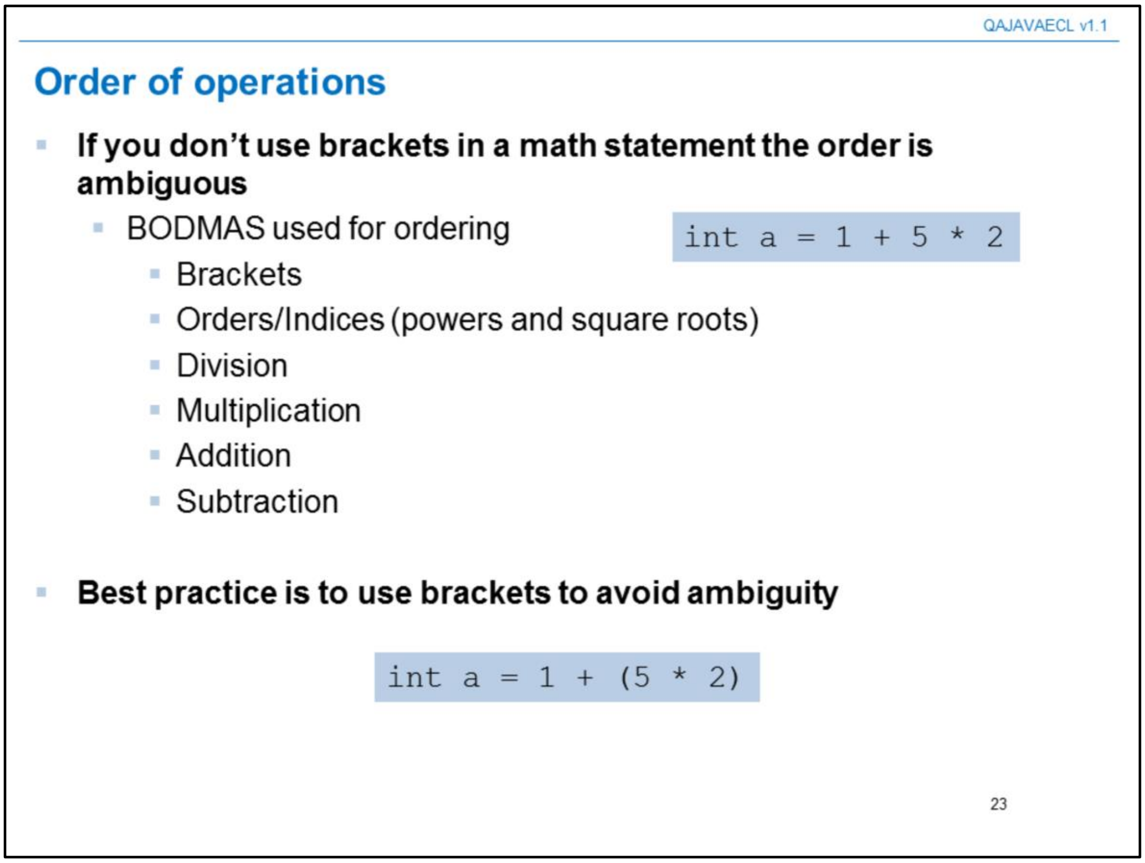
21



## Comparisons

| | |
|---|---|
| x > y | X is greater than Y |
| x < y | X is less than Y |
| x >= y | X is greater than or equal to Y |
| x <= y | X is less than or equal to Y |
| x != y | X is not equal to Y |
| x == y | X is equal to Y |

- **All return a true or a false value based on the numbers given**
- **Make sure you use two equals signs when comparing if x is equal to y. Otherwise it is an assignment statement!**



## Order of operations

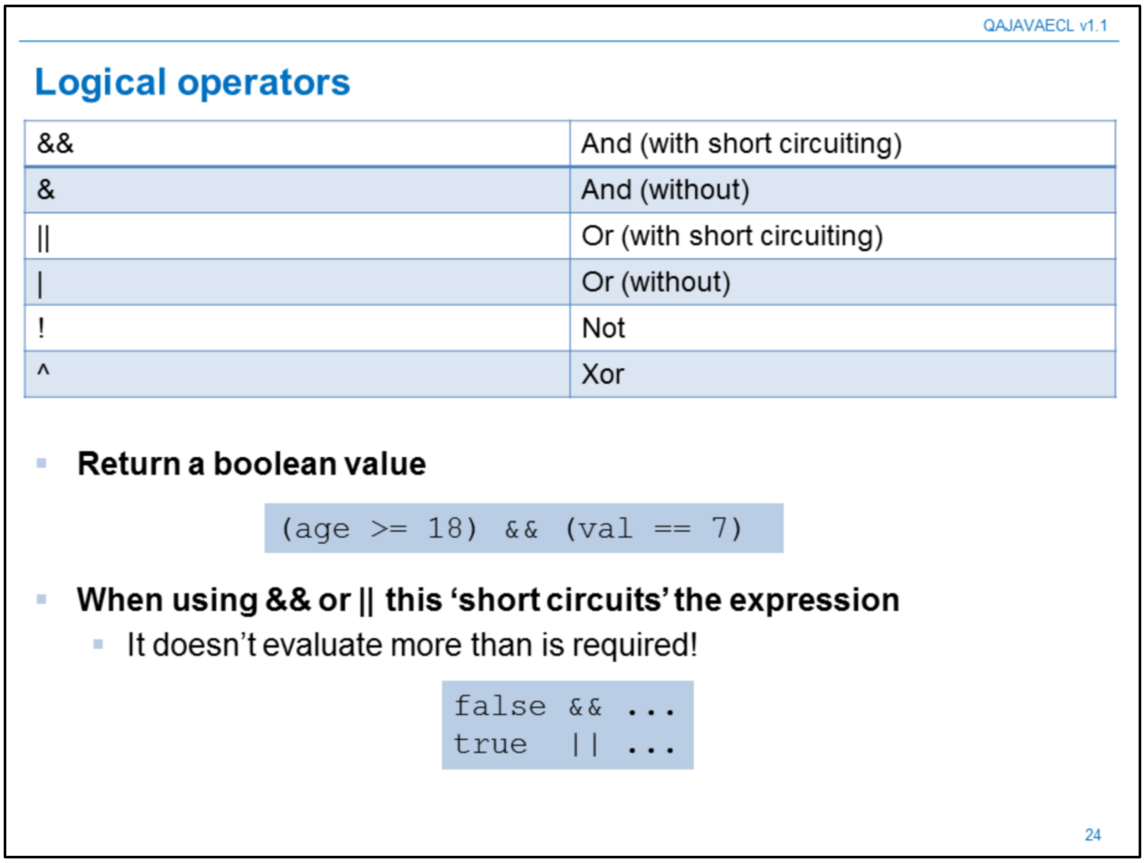
- **If you don't use brackets in a math statement the order is ambiguous**
  - BODMAS used for ordering
    - Brackets
    - Orders/Indices (powers and square roots)
    - Division
    - Multiplication
    - Addition
    - Subtraction

```
int a = 1 + 5 * 2
```

- **Best practice is to use brackets to avoid ambiguity**

```
int a = 1 + (5 * 2)
```

23



QAJAVAECL v1.1

## Logical operators

| | |
|---|---|
| && | And (with short circuiting) |
| & | And (without) |
| \|\| | Or (with short circuiting) |
| \| | Or (without) |
| ! | Not |
| ^ | Xor |

- **Return a boolean value**

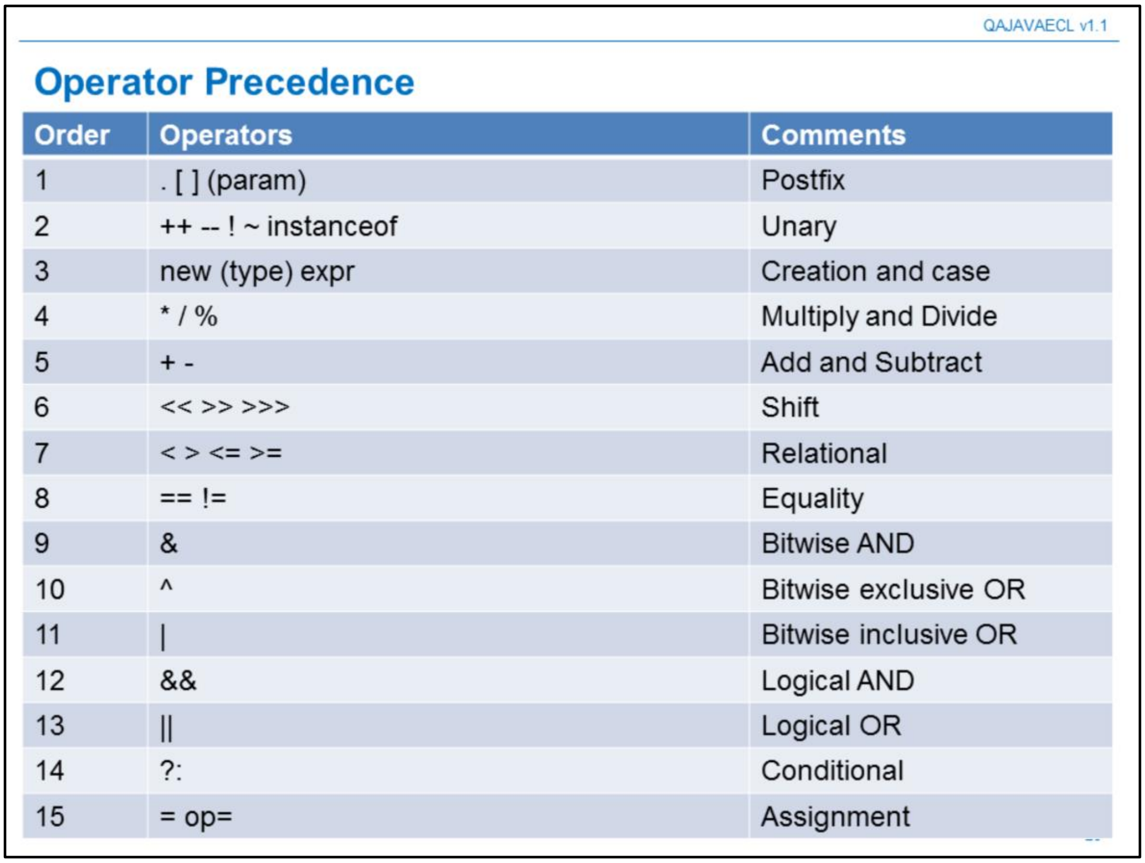
```
(age >= 18) && (val == 7)
```

- **When using && or || this 'short circuits' the expression**
  - It doesn't evaluate more than is required!

```
false && ...
true  || ...
```

24

Using one & or | symbol also can be used for bitwise comparisons. It does a logical and / or on the bits that make up a number. So 3 & 4 would compile, whereas 3 && 4 would not.



QAJAVAECL v1.1

## Operator Precedence

| Order | Operators | Comments |
|---|---|---|
| 1 | . [ ] (param) | Postfix |
| 2 | ++ -- ! ~ instanceof | Unary |
| 3 | new (type) expr | Creation and case |
| 4 | * / % | Multiply and Divide |
| 5 | + - | Add and Subtract |
| 6 | << >> >>> | Shift |
| 7 | < > <= >= | Relational |
| 8 | == != | Equality |
| 9 | & | Bitwise AND |
| 10 | ^ | Bitwise exclusive OR |
| 11 | \| | Bitwise inclusive OR |
| 12 | && | Logical AND |
| 13 | \|\| | Logical OR |
| 14 | ?: | Conditional |
| 15 | = op= | Assignment |

Precedence refers to the order in which operators are executed. For example, multiplication is always performed before addition or subtraction. The table on the slide shows all the Java operators in order of precedence (i.e. 1 is the highest). With the exception of the unary and assignment operators, which are right associative, operators with the same precedence are executed from left to right.

The tertiary operator ? Is used in the form: **x = (y > 0) ? 50 : -50**
This operation says that if y is greater than zero, x should be set to 50. Otherwise (if y is less than or equal to zero, set x to be -50.

The operators ( ~ , & , | , ^ ) operate on the binary representation of a number.

~ acts on a single number and inverts all bits within that number - so 101 would become 010.

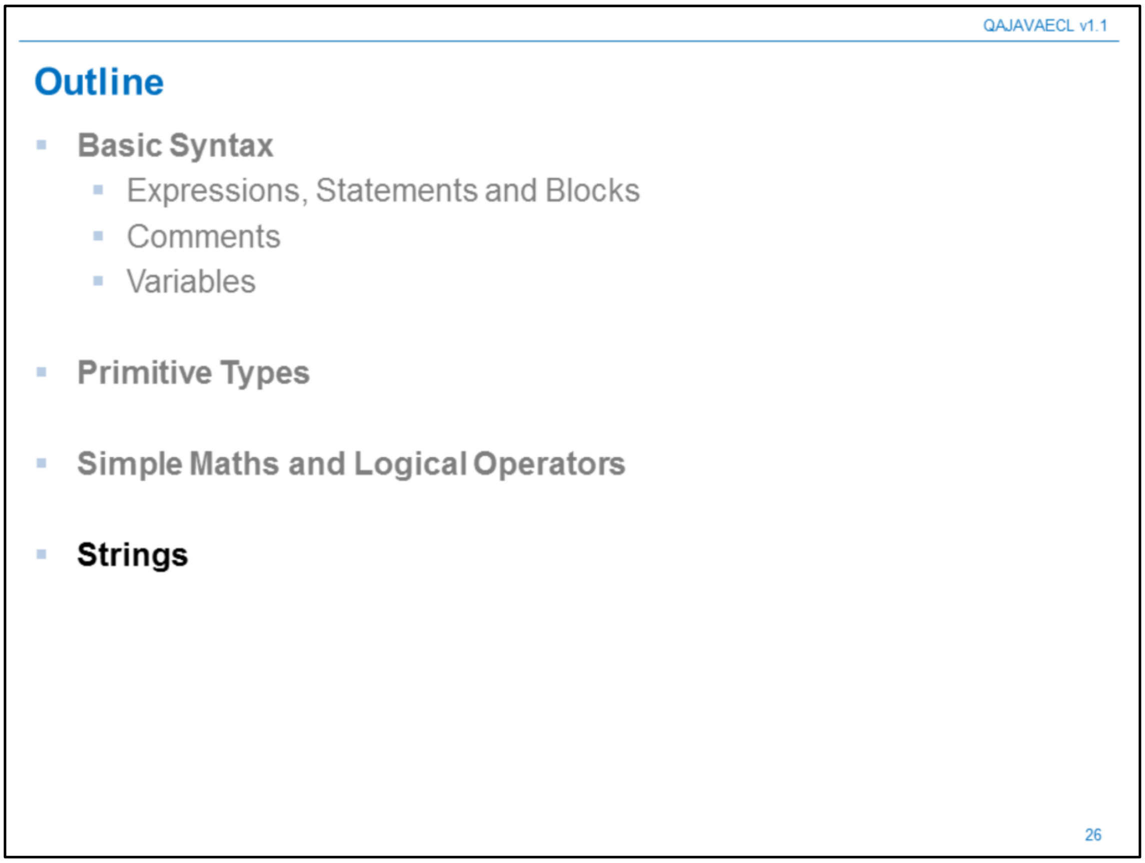
& compares the bits within two numbers - if both bits are set to 1 the result is 1, otherwise 0

1100 & 1010 would give 1000

| compares the bits within two numbers - if either is set to 1 the result is 1, otherwise 0

1100 | 1010 would give 1110

^ compares the bits within two numbers - where one is set to 1 and the other 0 the result is 1, but if both bits are the same (either both 1 or both 0) the result is 0 - i.e. the test is that of "exclusive or".

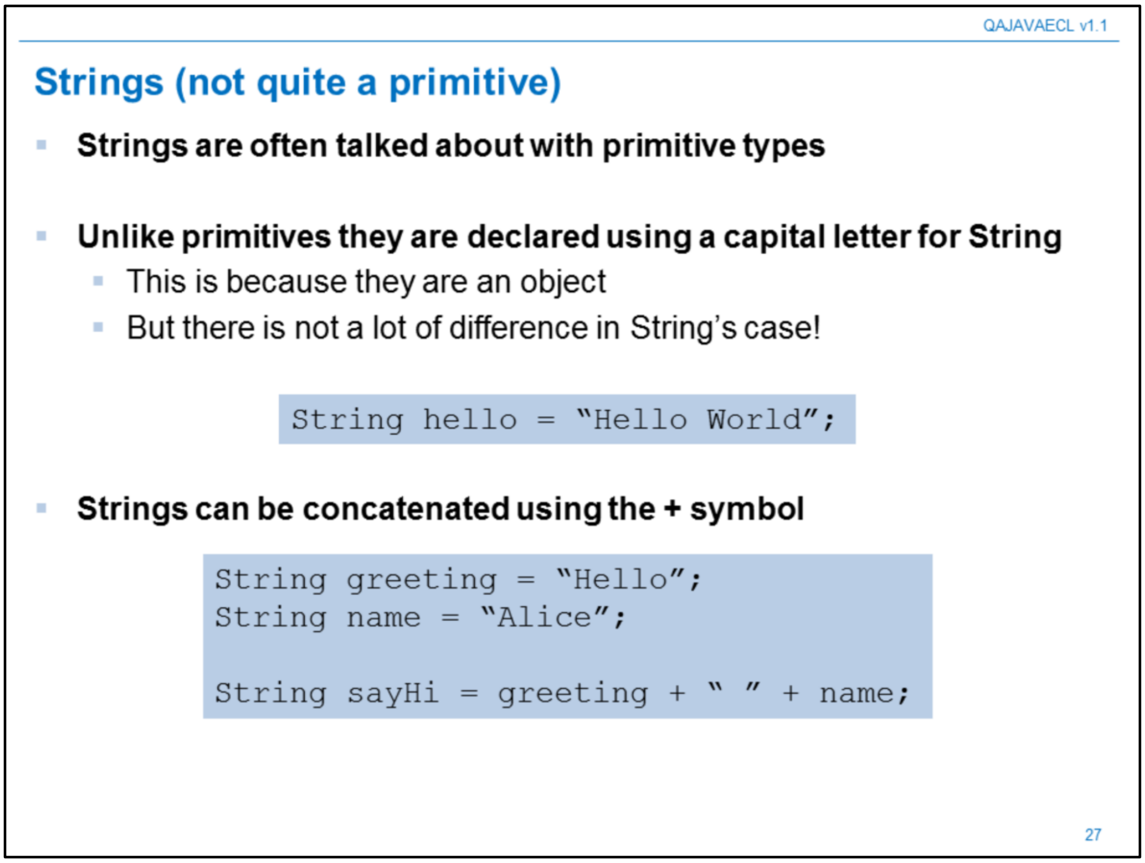


QAJAVAECL v1.1

## Outline

- **Basic Syntax**
  - Expressions, Statements and Blocks
  - Comments
  - Variables
- **Primitive Types**
- **Simple Maths and Logical Operators**
- **Strings**

26



## Strings (not quite a primitive)

- **Strings are often talked about with primitive types**
- **Unlike primitives they are declared using a capital letter for String**
  - This is because they are an object
  - But there is not a lot of difference in String’s case!

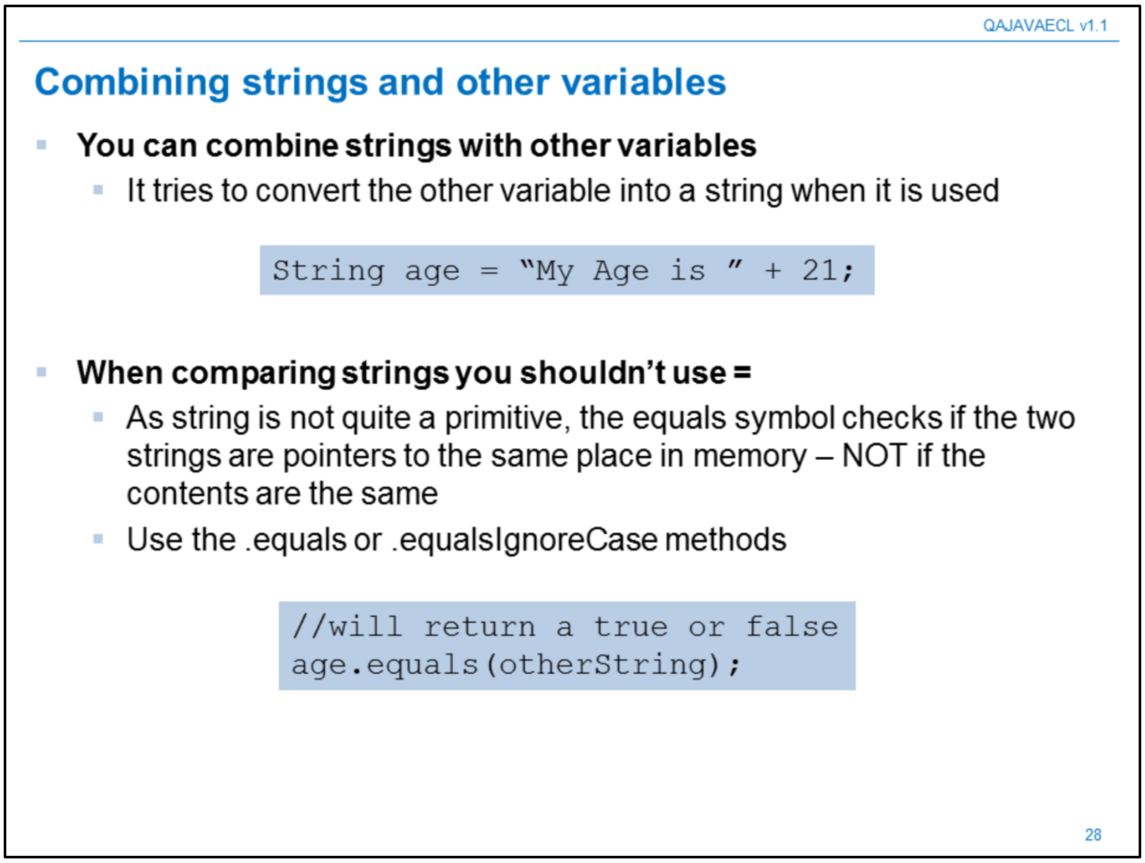
```
String hello = “Hello World”;
```

- **Strings can be concatenated using the + symbol**

```
String greeting = “Hello”;
String name = “Alice”;

String sayHi = greeting + “ ” + name;
```

27



QAJAVAECL v1.1

## Combining strings and other variables

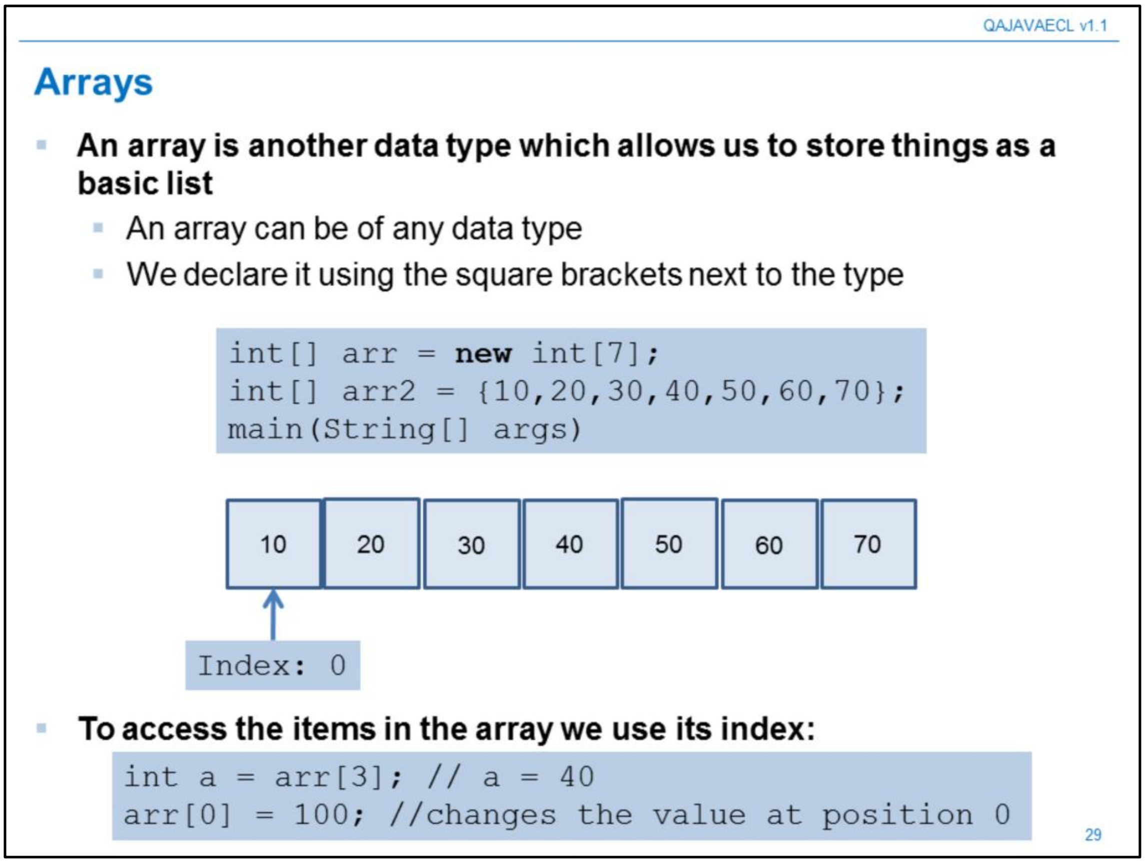
- **You can combine strings with other variables**
  - It tries to convert the other variable into a string when it is used

```
String age = "My Age is " + 21;
```

- **When comparing strings you shouldn't use =**
  - As string is not quite a primitive, the equals symbol checks if the two strings are pointers to the same place in memory – NOT if the contents are the same
  - Use the .equals or .equalsIgnoreCase methods

```
//will return a true or false
age.equals(otherString);
```

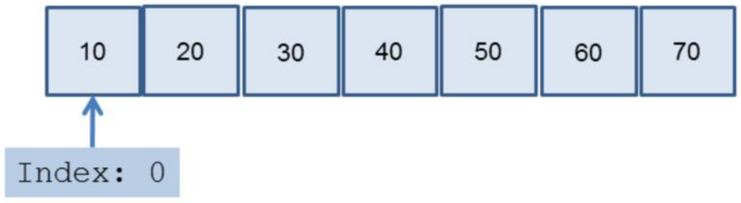
28



## Arrays

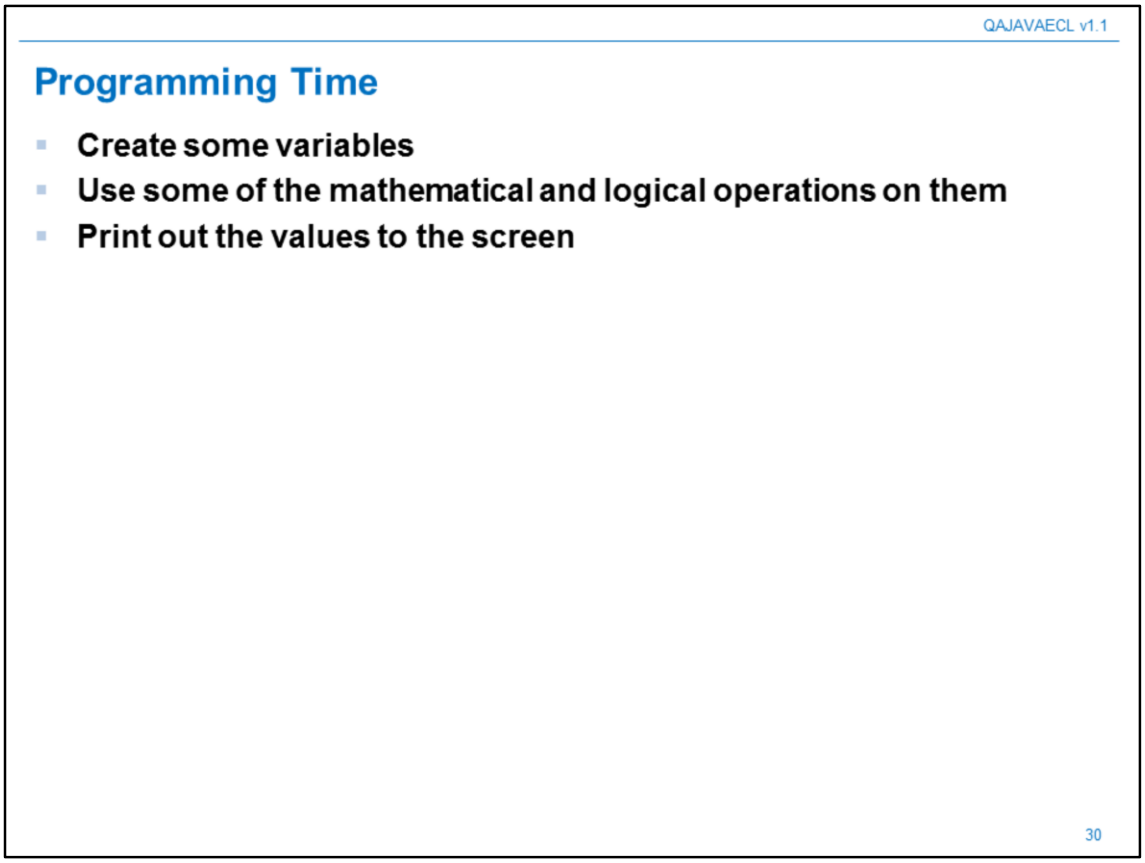
- **An array is another data type which allows us to store things as a basic list**
  - An array can be of any data type
  - We declare it using the square brackets next to the type

```
int[] arr = new int[7];
int[] arr2 = {10,20,30,40,50,60,70};
main(String[] args)
```



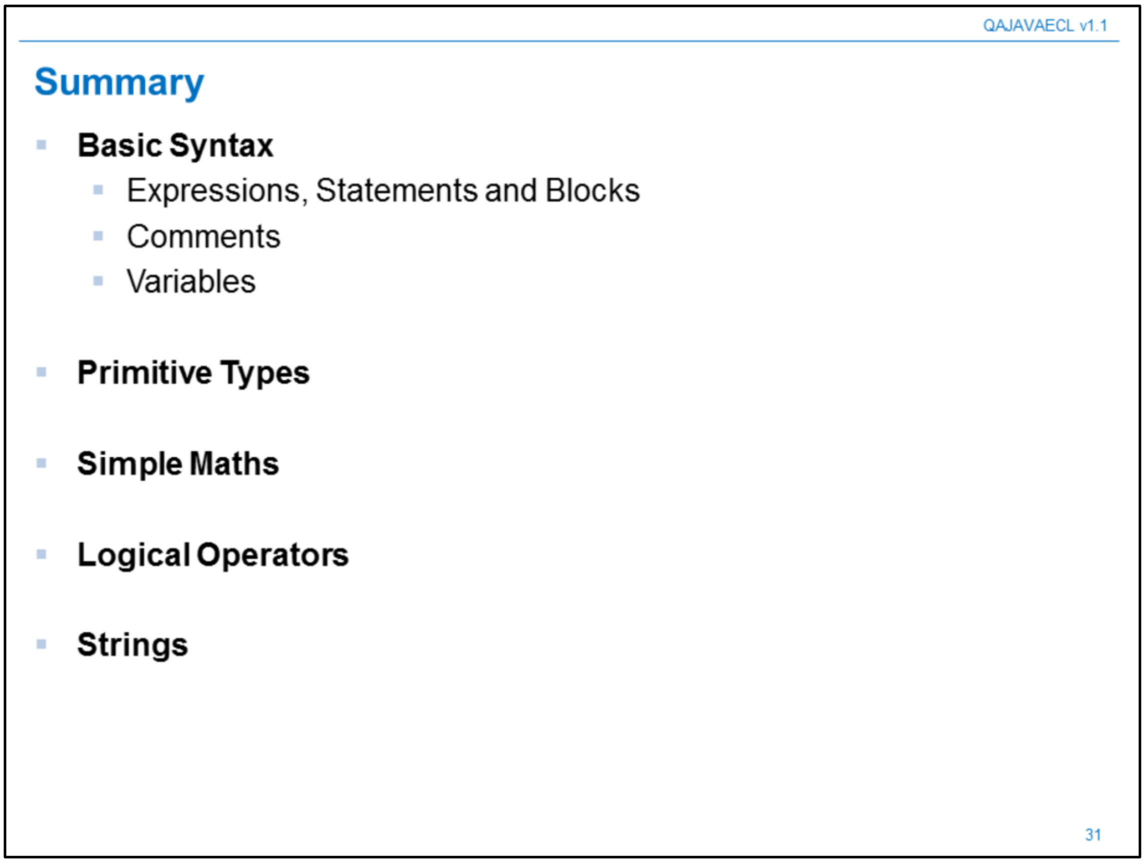
- **To access the items in the array we use its index:**

```
int a = arr[3]; // a = 40
arr[0] = 100; //changes the value at position 0
```



## Programming Time

- Create some variables
- Use some of the mathematical and logical operations on them
- Print out the values to the screen



QAJAVAECL v1.1

## Summary

- **Basic Syntax**
  - Expressions, Statements and Blocks
  - Comments
  - Variables
- **Primitive Types**
- **Simple Maths**
- **Logical Operators**
- **Strings**

31