

Java's basic constructs

Statements

A *statement* is the smallest unit of code that can be executed as a stand-alone instruction.

Example.

```
System.out.print("Hello, World!");
```

Statements

A *statement* is the smallest unit of code that can be executed as a stand-alone instruction.

Example.

```
double f = (9.0/5.0)*c + 32;
```

Expressions

An *expression* is a code fragment that has a value.

Example.

1823

Expressions

An *expression* is a code fragment that has a value.

Example.

$1823 + n$ (assuming n has a value)

Expressions

An *expression* is a code fragment that has a value.

Example.

`"Trinity College"`

Expressions

An *expression* is a code fragment that has a value.

Non example.

```
System.out.print("Hello, World!");
```

Expressions

An *expression* is a code fragment that has a value.

Non example.

```
double f = (9.0/5.0)*c + 32;
```

Statement and expressions

An *expression* is usually part of a statement.

Example. This statement

```
System.out.print("Hello, World!");
```

contains an expression "Hello, World!".

Statement and expressions

An *expression* is usually part of a statement.

Example. This statement

```
double f = (9.0/5.0) * c + 32;
```

contains an expression `9.0/5.0`.

Types

Each value is equipped with its *type*.

Since an expression has a value, each expression is also equipped with its type (namely, the type of the value of an expression).

Primitive data types

In Java, there are four primitive data types:

- **int**—integers: `...`, `-1`, `0`, `1`, `2`, `...`
- **float, double**—reals: `...`, `0.0`, `...`
- **char**—letters: `'a'`, `'b'`, `'c'`, `...`
- **boolean**—boolean values: `true`, `false`

Statement blocks

A *statement block* is a set of statements defined under a single construct (such as a class, a decision structure, a repetition structure, etc.).

Statement blocks

In general, in Java, a statement block begins with `{` and ends with `}`.

Note. While not required for compilation, you should make indentation whenever you start a new statement block after `{`.

Statement blocks

Example.

```
public class HelloWorld {  
    static public void main(String args[]) {  
        System.out.print("Hello, World!");  
    }  
}
```

Local variables

A *variable* is a name, or symbol, that represents a value that may be changed.

A variable must be defined with a type. It may be used multiple times, but it can be defined only once.

A *local variable* is a variable defined within a statement block. Its scope is limited to inside the block.

Local variables

Example.

```
{  
    int i = 10;  
  
    i = i + 1;  
    System.out.print(i);  
}
```

Local variables

Example.

```
{  
    int c = 100;  
    double f = (9.0/5.0)*c + 32;  
  
    System.out.print(f + " degrees.");  
}
```

Strings

Strings

In Java, a string is technically an *object* defined by the class `String`. However, in many ways, strings act like regular variables.

A string value must be specified by a pair of double quotations.

Example.

```
String s = "Trinity";
```

Characters

A character, on the other hand, is one of the four primitive data types: `boolean`, `char`, `float/double` and `int`.

A character value must be specified by a pair of single quotations.

Example.

```
char c = 'T';
```

Strings

Each letter of a string is a character indexed by an integer, starting from 0.

Example.

0	1	2	3	4	5	6
T	r	i	n	i	t	y

Strings

To access a string `s`'s `i`th character, we must invoke `charAt()`.

Example.

```
String s = "Trinity";
```

```
int i = 5;
```

```
System.out.print(s.charAt(i));
```

Output: `t` (as a `char` value)

Strings

Strings are *immutable*. Unlike variables, characters inside a string cannot be modified.

Example. This does not work.

```
String s = "Trinity";  
s.charAt(0) = 't';
```

Strings

Strings can be concatenated with +.

Example.

```
String s = "Trinity";  
s = s + "College";  
System.out.print(s);
```

Output: TrinityCollege