Basic datatypes in Java

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 2

Scalar types

- In an object-oriented language, all data should be encapsulated as objects

Scalar types

- In an object-oriented language, all data should be encapsulated as objects
- However, this is cumbersome
  - Useful to manipulate numeric values like conventional languages

Scalar types

- In an object-oriented language, all data should be encapsulated as objects
- However, this is cumbersome
  - Useful to manipulate numeric values like conventional languages
- Java has eight primitive scalar types
  - `int`, `long`, `short`, `byte`
  - `float`, `double`
  - `char`
  - `boolean`

Scalar types

- In an object-oriented language, all data should be encapsulated as objects
- However, this is cumbersome
  - Useful to manipulate numeric values like conventional languages
- Java has eight primitive scalar types
  - `int`, `long`, `short`, `byte`
  - `float`, `double`
  - `char`
  - `boolean`
- Size of each type is fixed by JVM
  - Does not depend on native architecture

Type	Size in bytes
`int`	4
`long`	8
`short`	2
`byte`	1
`float`	4
`double`	8
`char`	2
`boolean`	1

Scalar types

- In an object-oriented language, all data should be encapsulated as objects
- However, this is cumbersome
  - Useful to manipulate numeric values like conventional languages
- Java has eight primitive scalar types
  - `int`, `long`, `short`, `byte`
  - `float`, `double`
  - `char`
  - `boolean`
- Size of each type is fixed by JVM
  - Does not depend on native architecture

Type	Size in bytes
`int`	4
`long`	8
`short`	2
`byte`	1
`float`	4
`double`	8
`char`	2
`boolean`	1

- 2-byte `char` for Unicode

Declarations, assigning values

- We declare variables before we use them

```
int x, y;
double y;
char c;
boolean b1, b2;
```

  - Note the semicolons after each statement

Declarations, assigning values

- We declare variables before we use them

```
int x, y;
double y;
char c;
boolean b1, b2;
```

  - Note the semicolons after each statement

- The assignment statement works as usual

```
int x,y;
x = 5;
y = 7;
```

Declarations, assigning values

- We declare variables before we use them

```
int x, y;
double y;
char c;
boolean b1, b2;
```

  - Note the semicolons after each statement

- The assignment statement works as usual

```
int x,y;
x = 5;
y = 7;
```

- Characters are written with single-quotes (only)

```
char c,d;

c = 'x';
d = '\u03C0'; // Greek pi, unicode
```

  - Double quotes denote strings

Declarations, assigning values

- We declare variables before we use them

```
int x, y;
double y;
char c;
boolean b1, b2;
```

  - Note the semicolons after each statement

- The assignment statement works as usual

```
int x,y;
x = 5;
y = 7;
```

- Characters are written with single-quotes (only)

```
char c,d;

c = 'x';
d = '\u03C0'; // Greek pi, unicode
```

  - Double quotes denote strings

- Boolean constants are `true`, `false`

```
boolean b1, b2;

b1 = false;
b2 = true;
```

Initialization, constants

- Declarations can come anywhere

```
int x;
x = 10;
double y;
```

  - Use this judiciously to retain readability

Initialization, constants

- Declarations can come anywhere

```
int x;
x = 10;
double y;
```

  - Use this judiciously to retain readability

- Initialize at time of declaration

```
int x = 10;
double y = 5.7;
```

Initialization, constants

- Declarations can come anywhere

```
int x;
x = 10;
double y;
```

  - Use this judiciously to retain readability

- Initialize at time of declaration

```
int x = 10;
double y = 5.7;
```

- Can we declare a value to be a constant?

```
float pi = 3.1415927f;

pi = 22/7;  // Disallow?
```

  - Note: Append `f` after number for `float`, else interpreted as `double`

Initialization, constants

- Declarations can come anywhere

```
int x;
x = 10;
double y;
```

  - Use this judiciously to retain readability

- Initialize at time of declaration

```
int x = 10;
double y = 5.7;
```

- Can we declare a value to be a constant?

```
float pi = 3.1415927f;

pi = 22/7;  // Disallow?
```

  - Note: Append `f` after number for `float`, else interpreted as `double`

- Modifier `final` indicates a constant

```
final float pi = 3.1415927f;

pi = 22/7;  // Flagged as error;
```

Operators, shortcuts, type casting

- Arithmetic operators are the usual ones
  - `+`, `-`, `*`, `/`, `%`

Operators, shortcuts, type casting

- Arithmetic operators are the usual ones
  - `+`, `-`, `*`, `/`, `%`
- No separate integer division operator `//`
- When both arguments are integer, `/` is integer division

```
float f = 22/7;    // Value is 3.0
```

  - Note implicit conversion from `int` to `float`

Operators, shortcuts, type casting

- Arithmetic operators are the usual ones
  - `+`, `-`, `*`, `/`, `%`
- No separate integer division operator `//`
- When both arguments are integer, `/` is integer division

```
float f = 22/7;    // Value is 3.0
```

  - Note implicit conversion from `int` to `float`
- No exponentiation operater, use `Math.pow()`
- `Math.pow(a,n)` returns a^n

Operators, shortcuts, type casting

- Arithmetic operators are the usual ones
  - `+`, `-`, `*`, `/`, `%`
- No separate integer division operator `//`
- When both arguments are integer, `/` is integer division

```
float f = 22/7;    // Value is 3.0
```

  - Note implicit conversion from `int` to `float`
- No exponentiation operater, use `Math.pow()`
- `Math.pow(a,n)` returns a^n
- Special operators for incrementing and decrementing integers

```
int a = 0, b = 10;
a++;    // Same as a = a+1
b--;    // Same as b = b-1
```

Operators, shortcuts, type casting

- Arithmetic operators are the usual ones
  - `+`, `-`, `*`, `/`, `%`
- No separate integer division operator `//`
- When both arguments are integer, `/` is integer division

```
float f = 22/7;    // Value is 3.0
```

  - Note implicit conversion from `int` to `float`
- No exponentiation operater, use `Math.pow()`
- `Math.pow(a,n)` returns a^n
- Special operators for incrementing and decrementing integers

```
int a = 0, b = 10;
a++;    // Same as a = a+1
b--;    // Same as b = b-1
```

- Shortcut for updating a variable

```
int a = 0, b = 10;
a += 7;     // Same as a = a+7
b *= 12;    // Same as b = b*12
```

Strings

- `String` is a built in class

```
String s,t;
```

Strings

- `String` is a built in class

```
String s,t;
```

- String constants enclosed in double quotes

```
String s = "Hello", t = "world";
```

Strings

- `String` is a built in class

```
String s,t;
```

- String constants enclosed in double quotes

```
String s = "Hello", t = "world";
```

- `+` is overloaded for string concatenation

```
String s = "Hello";
String t = "world";
String u = s + " " + t;
  // "Hello world"
```

Strings

- `String` is a built in class

```
String s,t;
```

- String constants enclosed in double quotes

```
String s = "Hello", t = "world";
```

- + is overloaded for string concatenation

```
String s = "Hello";
String t = "world";
String u = s + " " + t;
  // "Hello world"
```

- Strings are not arrays of characters
  - Cannot write

```
s[3] = 'p';
s[4] = '!';
```

Strings

- `String` is a built in class

```
String s,t;
```

- String constants enclosed in double quotes

```
String s = "Hello", t = "world";
```

- + is overloaded for string concatenation

```
String s = "Hello";
String t = "world";
String u = s + " " + t;
  // "Hello world"
```

- Strings are not arrays of characters
  - Cannot write

```
s[3] = 'p';
s[4] = '!';
```

- Instead, invoke method `substring` in class `String`
  - `s = s.substring(0,3) + "p!";`

Strings

- `String` is a built in class

```
String s,t;
```

- String constants enclosed in double quotes

```
String s = "Hello", t = "world";
```

- + is overloaded for string concatenation

```
String s = "Hello";
String t = "world";
String u = s + " " + t;
  // "Hello world"
```

- Strings are not arrays of characters
  - Cannot write

```
s[3] = 'p';
s[4] = '!';
```

- Instead, invoke method `substring` in class `String`
  - `s = s.substring(0,3) + "p!";`
- If we change a `String`, we get a new object
  - After the update, `s` points to a new `String`
  - Java does automatic garbage collection

Arrays

- Arrays are also objects

Arrays

- Arrays are also objects
- Typical declaration

```
int[] a;
a = new int[100];
```

  - Or `int a[]` instead of `int[] a`
  - Combine as `int[] a = new int[100];`

Arrays

- Arrays are also objects
- Typical declaration

```
int[] a;
a = new int[100];
```

  - Or `int a[]` instead of `int[] a`
  - Combine as `int[] a = new int[100];`
- `a.length` gives size of `a`
  - Note, for `String`, it is a method `s.length()`!

Arrays

- Arrays are also objects
- Typical declaration

```
int[] a;
a = new int[100];
```

  - Or `int a[]` instead of `int[] a`
  - Combine as `int[] a = new int[100];`
- `a.length` gives size of `a`
  - Note, for `String`, it is a method `s.length()`!
- Array indices run from `0` to `a.length-1`

Arrays

- Arrays are also objects
- Typical declaration

```
int[] a;
a = new int[100];
```

  - Or `int a[]` instead of `int[] a`
  - Combine as `int[] a = new int[100];`
- `a.length` gives size of `a`
  - Note, for `String`, it is a method `s.length()`!
- Array indices run from `0` to `a.length-1`
- Size of the array can vary

Arrays

- Arrays are also objects
- Typical declaration

```
int[] a;
a = new int[100];
```

  - Or `int a[]` instead of `int[] a`
  - Combine as `int[] a = new int[100];`
- `a.length` gives size of `a`
  - Note, for `String`, it is a method `s.length()`!
- Array indices run from `0` to `a.length-1`
- Size of the array can vary
- Array constants: `{v1, v2, v3}`

Arrays

- Arrays are also objects
- Typical declaration

```
int[] a;
a = new int[100];
```

  - Or `int a[]` instead of `int[] a`
  - Combine as `int[] a = new int[100];`
- `a.length` gives size of `a`
  - Note, for `String`, it is a method `s.length()`!
- Array indices run from `0` to `a.length-1`
- Size of the array can vary
- Array constants: `{v1, v2, v3}`
- For example

```
int[] a;
int n;

n = 10;
a = new int[n];

n = 20;
a = new int[n];

a = {2, 3, 5, 7, 11};
```

Summary

- Java allows scalar types, which are not objects
  - `int`, `long`, `short`, `byte`, `float`, `double`, `char`, `boolean`
- Declarations can include initializations
- Strings and arrays are objects
- Numerous versions of Java: we will use Java 11
- Extensive online documentation — look up in case of doubt
  `https://docs.oracle.com/en/java/javase/11/docs/api/index.html`