Java Variables

Josh Wilcox (jw14g24@soton.ac.uk)

January 28, 2025

Contents

1 Variables

- Variables store objects
- Either builtin types like ints and bools
- But can also store objects that you create in the program

1.1 Naming Principles

- Do not start variable names with capital letters or numbers
  - Capital Letters should be resulting for classes and constants
- Make variable names meaningful

2 Variable Types

2.1 Primitive Types

- In Java there are 8 primitive types
- **Integers**
  - Byte - 8 bits
  - Short - 16 bits
  - Int - 32 bits
  - Long - 64 bits
- **Decimals**
  - Float - 32 bits
  - Double - 64 bits
- **Others**
  - Char - 16 bits
  - Boolean - true / false - **LOWER CASE**

2.2 Properties

- Defined within Java - which can be used directly
- Integral part of the Java language
- Stored directly in variables
- *Pass by copy*
  - When a primitive variable is passed to a method, a copy of the variable's value is passed
  - Changes to the parameter within the method do not affect the original variable

```java
int a;
a = 10;
int b;
b = 5;
int c;
c = a;
c = 2*c;
b = a*c;
```

    * In this example, when `c` is assigned the value of `a` (`c = a;`), a copy of the value of `a` is made and stored in `c`.

- ∗ Changes to `c` (such as `c = 2*c;`) do not affect the value of `a`.
- ∗ Similarly, when `b` is assigned the value of `a * c` (`b = a*c;`), the values of `a` and `c` are copied and used in the calculation, without modifying `a` or `c`.

2.3 Objects

- Defined with a class
- Default value of a class is `null`

2.3.1 Properties

- Defined in classes
  - Either written by the programmer
  - Or provided by a library
- Stored indirectly
  - Variables *reference* the objects - Like a pointer
  - The objects themselves are stored in memory
- Pass by reference

```
Elephant a;
a = new Elephant();
Elephant b = new Elephant();
Elephant c;
c = a;
a = b;
c = null;
```

  - In this example, when `c` is assigned the value of `a` (`c = a;`), both `c` and `a` reference the same `Elephant` object.
  - Assigning `a = b;` makes `a` reference a different `Elephant` object, while `c` still references the original `Elephant`.
  - Setting `c = null;` removes the reference from `c` to the original `Elephant` object.

3 Scopes

- The rule of thumb is that a variable can be seen anywhere within the pair of curly braces it is **declared in**

```java
public class Account{
    int balance = 100;
    public void withdrawFiver(){
        balance = balance - 5;
    }
    public void withdrawTenner(){
        int tenner = 10;
        balance = balance - tenner;
    }
    public void withdrawFifty(){
        balance = balance - (tenner * 5);
    }
    public void closeAccount(){
        int balance = 0;
    }
}
```

- `balance` is a member variable of the `Account` class
- As a member variable of the class, it is visible by every sub-method within the class
- The `tenner` variable in the `withdrawTenner()` method is a local variable and only exists for the duration of the method in which it's declared
- This means that the `withdrawFifty()` method would not compile as it tries to access a variable that has not been defined
- The `closeAccount()` method will cause a conflict as balance has been declared twice
  - A local variable in `closeAccount()`
  - A member variable in the `Account` class
  - Any modifications to `balance` in `closeAccount()` will default to the local version and the member variable in the overarching class will remain unchanged

3.1 *This* Keyword

- To reference a member variable within a method in a class we should use the `this` keyword

```java
public class Account{
    int money = 100;
    public void withdraw(int money){
        money = money - money; // All money variables in withdraw() are local, so it will always = 0
    }
}
```

```java
public class Account{
    int money = 100;
    public void withdraw(int money){
        this.money = this.money - money; // Actually changes the true money member variable
    }
}
```