Static Variables

Josh Wilcox (jw14g24)

May 3, 2025

Table of Contents

Overview

Overview

- A static instance variable in the class is **shared across all instances of a class.**
- It allows for a single copy of the variable to be used by all instances, which can be useful for maintaining state or configuration that is common to all instances.
- Static variables can also be used to implement constants that are shared across instances, ensuring that the value remains consistent throughout the application.
- Additionally, static variables can improve performance by reducing memory usage when multiple instances of a class are created.
- Static variables are initialized only once, at the start of the program, which can lead to faster access times compared to instance variables.
- They can also be used to manage resources more efficiently, as they allow for a centralized control over shared data.

Think of static variables as **class variables** - They are not just properties of an Object instance, but are actually a property of the **class** and therefore all instances of it

- Building on this, you can just use the Class name to update the static variable instead of the objects
- For example do `Game.version = 5` instead of `hangman.version = 5`

Initialisation of Static Variables

JVM details

- The `.class` file is given into the class loader subsystem
- Static variables are stored in the **heap**
  - There is a **constant pool** in the heap
  - Remember, the **heap** is shared between all threads in a Java program
- Non-Static variables are stored in **stacks** on a per-thread basis

Starting a Program

- Before the program is even run - Objects are constructed etc. - Static variables are stored somewhere in memory
  - Static Methods in the Metaspace
  - Static Variables in the **Heap**
- Initialisation code for static fields get executed at **class** object construction time
- Initialisation code for non-static fields get executed at **object** construction time

`this` Keyword

- The `this` keyword refers to the current instance of a class
- Static variables belong to the class itself, not any specific instance
- Therefore, `this` cannot be used with static variables because:
  - Static methods/variables can be accessed without creating an object
  - No instance exists when static members are initialized
  - Static context has no access to instance-specific information
- Attempting to use `this` in a static context will result in a compilation error

Static Final variables (Constants)

Overview

- `static final` variables **must be declared directly under the first line of the class**

```
1  public class Game {
2      static final version = 3; // Must be declared here!!!
3      public Game(){
4          // static finals can not be declared in the constructor
5      }
6  }
```

- Static Finals are more commonly known as **constants** - Like constant variables declared at the very beginning of python files

Static methods

Overview

- A static method is also **class-based** instead of **object-based**
- This means, like with static variables, a static method can not reference variables or methods that are unique to an object
  - Static methods can't depend on values or fields stored in a particular instance
  - Static methods also can't use the keyword `this`
- Static Methods can only depend on Static Fields!

Why use Static Methods

- Sometimes you don't need to make instances of classes
  - For example - `Arrays.asList()` - `Arrays` is the class name and not an object name
  - This is also really common with the `Math` class such as `Math.min()`
    - The constructor of `Math` is private!