# Safe Initialization

## Final Keyword & Java Memory Model

# final Keyword

final can be used with class, variable, method

(1) Restrict changing value of variable

(2) Restrict method overriding

(3) Restrict inheritance

# Final Keyword

- Create unmodifiable references
- Restrict method overriding
- Restrict class inheritence

- Part of Java Memory Model
- Guarantees a field visibility in a multi-threaded application
- Safe initialization for objects, arrays and collections

```java
public class NonFinalVariable {

    public Integer value;

    public NonFinalVariable() {
        this.value = 1;
    }
}
```

```java
public class FinalVariable {

    public final Integer value;

    public FinalVariable() {
        this.value = 1;
    }
}
```

```java
public class FinalVariable {

    public final int[] value;

    public FinalVariable() {
        this.value = new int[5];
        this.value[0] = 1;
        this.value[1] = 2;
        this.value[2] = 3;
    }
}
```

```java
public class FinalVariable {

    public final List<Integer> value;

    public FinalVariable() {
        this.value = new ArrayList<>();
        this.value.add(1);
        this.value.add(2);
        this.value.add(3);
    }
}
```

```java
public class FinalVariable {

    public final InnerClass value;

    public FinalVariable() {
        this.value = new InnerClass( val: 1);
    }

    private static class InnerClass {
        public int innerValue;

        public InnerClass(int val) {
            this.innerValue = val;
        }
    }
}
```

```java
public class PartiallyFinalVariables {

    public final int value1;
    public int value2;
    public int value3;

    public PartiallyFinalVariables() {
        this.value1 = 1;
        this.value2 = 2;
        this.value3 = 3;
    }
}
```

```java
public class FinalVariables {

    public final int value1;
    public final int value2;
    public final int value3;

    public FinalVariables() {
        this.value1 = 1;
        this.value2 = 2;
        this.value3 = 3;
    }
}
```

```java
public class Initialization {

   private Object obj1 = new Object();
   private Object obj2;
   private Object obj3;

   {
      this.obj2 = new Object();
   }

   public Initialization() {
      this.obj3 = new Object();
   }
}
```

```
public pbouda.bytecode.examples.Initialization();
  descriptor: ()V
  flags: ACC_PUBLIC
  Code:
    stack=3, locals=1, args_size=1
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: aload_0
       5: new           #2                  // class java/lang/Object
       8: dup
       9: invokespecial #1                  // Method java/lang/Object."<init>":()V
      12: putfield      #3                  // Field obj1:Ljava/lang/Object;
      15: aload_0
      16: new           #2                  // class java/lang/Object
      19: dup
      20: invokespecial #1                  // Method java/lang/Object."<init>":()V
      23: putfield      #4                  // Field obj2:Ljava/lang/Object;
      26: aload_0
      27: new           #2                  // class java/lang/Object
      30: dup
      31: invokespecial #1                  // Method java/lang/Object."<init>":()V
      34: putfield      #5                  // Field obj3:Ljava/lang/Object;
      37: return
```

# << **jcstress demo** >>

http://openjdk.java.net/projects/code-tools/jcstress/

# !! I can’t use final !!

- Synchronized block (implicit locks)
- Explicit locks, volatile keyword
- CAS operations – Atomic*, *Adder
- java.util.concurrent package
  - SynchronizedMap, ConcurrenHashMap
  - CopyOnWriteArray(List|Set),
  - Synchronized(List|Set)
  - BlockingQueue, ConcurrentLinkedQueue
- VarHandles, ? Unsafe ?

# Summary

Always start with a FINAL variable and change it only if the object’s state is really supposed to be mutable.

.. then you need to handle visibility problems using memory barriers generated by LOCKS or VOLATILE.