Understanding Variable Rules and Scope

Fields, local variables, primitives, and reference types

Richard Monson-Haefel
Sr. Software Engineer

www.monsonhaefel.com

Overview

- Fields, local variables, primitives, and reference types
- Members, access modifiers, and scope
- Method Variable Scope
- Variable naming rules and conventions
- The var variable
- The varargs parameter

Fields vs Local Variables

```
public class FieldsVsVariables {

    static int field_a;
    static String field_b;

    float field_c;
    String field_d;

    public void method_1( ){


        short variable_a;
        String varaible_b;

    }

    public static void main(String [] args){

        long variable_c;
        String varaible_d;

    }
}
```

Fields

- Class level
- Static (aka class) variable
- Instance variable
- Initialized by default

Type	Size	Default
boolean	1 bit	False
Char	16 bit	'\u0000'
Byte	8 bits	0
Short	16 bits	0
Int	32 bits	0
Long	64 bits	0
Float	32 bits	0.0
Double	64 bits	0.0

Fields vs Local Variables

```
public class FieldsVsVariables {

    static int field_a;
    static String field_b;

    float field_c;
    String field_d;

    public void method_1( ){

        short variable_a;
        String varaible_b;

    }

    public static void main(String [] args){

        long variable_c;
        String varaible_d;

    }
}
```

Fields

- Class level
- Static (aka class) variable
- Instance variable
- Initialized by default

Local variables

- Methods or code blocks
- Never static

Fields vs Local Variables

```
public class FieldsVsVariables {

    static int field_a;
    static String field_b;

    float field_c;
    String field_d;

    public void method_1( ){

        short variable_a;
        String varaible_b;

    }

    public static void main(String [] args){

        long variable_c;
        String varaible_d;

    }
}
```

Fields

- Class level
- Static (aka class) variable
- Instance variable
- Initialized by default

Local variables

- Methods or code blocks
- Never static
- Must be initialized

```java
public class FieldsVsVariables {

    static int field_a = 1000;
    static String field_b = "static string";

    float field_c = 3.14f;
    String field_d = "a instance string";

    public void method_1( ){

        short variable_a = 122;
        String varaible_b = "a string";

    }

    public static void main(String [] args){

        long variable_c = 122L;
        String varaible_d = "another string";

    }
}
```

Primitive fields and local variables Store values

name	values
field_a	1000
field_c	3.14f
variable_a	122
variable_c	122L

Object fields and local variables Store references

name	reference
field_b	0001
field_d	1010
variable_b	1100
variable_d	1110

Object values
"static string"
"an instance string"
"a string"
"another string"

```java
public class FieldsVsVariables {

    static int field_a = 1000;
    static String field_b = “static string”;

    float field_c = 3.14f;
    String field_d = “a instance string”;

    public void method_1( ){

        short variable_a = 122;
        String varaible_b = “a string”;

    }

    public static void main(String [] args){

        long variable_c = 122L;
        String varaible_d = “another string”;

    }
}
```

Primitive fields and local variables Store values

name	values
field_a	1000
field_c	3.14f
variable_a	122
variable_c	122L

Object fields and local variables Store references

name	reference	Object values
field_b	0001	“static string”
field_d	1010	“an instance string”
variable_b	1100	“a string”
variable_d	1110	“another string”

```java
public class FieldsVsVariables {

    static int field_a = 1000;
    static String field_b = “static string”;

    float field_c = 3.14f;
    String field_d = “a instance string”;

    public void method_1( ){

        short variable_a = 122;
        String varaible_b = “a string”;

    }

    public static void main(String [] args){

        long variable_c = 122L;
        String varaible_d = “another string”;

    }
}
```

Primitive fields and local variables Store values

name	values
field_a	1000
field_c	3.14f
variable_a	122
variable_c	122L

Object fields and local variables Store references

name	reference	Object values
field_a	0001	“static string”
field_d	1010	“an instance string”
variable_b	1100	“a string”
variable_d	1110	“another string”

```java
public class FieldsVsVariables {

    static int field_a = 1000;
    static String field_b = null;

    float field_c = 3.14f;
    String field_d = null;

    public void method_1( ){


        short variable_a = 122;
        String varaible_b = null;

    }

    public static void main(String [] args){

        long variable_c = 122L;
        String varaible_d = null;

    }
}
```

Primitive fields and local variables Store values

name	values
field_a	1000
field_c	3.14f
variable_a	122
variable_c	122L

Object fields and local variables Store references

name	reference
field_b	null
field_d	null
variable_b	null
variable_d	null

```java
public class FieldsVsVariables {

    static int field_a;
    static String field_b;

    float field_c;
    String field_d;

    public void method_1( ){

        short variable_a = 122;
        String varaible_b = “a string”;

    }

    public static void main(String [] args){

        long variable_c = 122L;
        String varaible_d = “another string”;

    }
}
```

Primitive fields and local variables Store values

name	values
field_a	0
field_c	0.0f
variable_a	122
variable_c	122L

Object fields and local variables Store references

name	reference	Object values
field_a	null	
field_d	null	
variable_b	1100	“a string”
variable_d	1110	“another string”

```java
public class FieldsVsVariables {

    static int field_a;
    static String field_b;

    float field_c;
    String field_d;

    public void method_1( ){

        short variable_a = 122;
        String varaible_b = “a string”;

    }

    public static void main(String [] args){

        long variable_c = 122L;
        String varaible_d = “another string”;

    }
}
```

Primitive fields and local variables Store values

name	values
field_a	0
field_c	0.0f
variable_a	122
variable_c	122L

Object fields and local variables Store references

name	reference	Object values
field_a	null	
field_d	null	
variable_b	1100	“a string”
variable_d	1110	“another string”

```java
public class FieldsVsVariables {

    static int field_a;
    static String field_b;

    float field_c;
    String field_d;

    public void method_1( ){

        short variable_a;
        String varaible_b;

    }

    public static void main(String [] args){

        long variable_c;
        String varaible_d;

    }
}
```

Primitive fields and local variables Store values

name	values
field_a	0
field_c	0.0f
variable_a	UD
variable_c	UD

Object fields and local variables Store references

name	reference
field_a	null
field_d	null
variable_b	UD
variable_d	UD

Members and Member Scope

Members

Static and instance fields and methods

```
public class MethodVariableScope {

    static { /*static initializer; not a member */ }

    { /* instance initializer; not a member*/ }

    // static (class) field; a member
    public static int staticField = 1;

    // instance field; a member
    public int instanceField = 1;

    // static (class) method; a member
    public void aStaticMethod(){ }

    // instance method; a member
    public void anInstanceMethod(){}

    // constructor; not a member
    public MethodVariableScope(){ }
}
```

Members

Static and instance fields and methods

```
public class MethodVariableScope {

    static { /*static initializer; not a member */ }

    { /* instance initializer; not a member*/ }

    // static (class) field; a member
    public static int staticField = 1;

    // instance field; a member
    public int instanceField = 1;

    // static (class) method; a member
    public void aStaticMethod(){ }

    // instance method; a member
    public void anInstanceMethod(){}

    // constructor; not a member
    public MethodVariableScope(){ }
}
```

Members

Members

Static and instance fields and methods

```
public class MethodVariableScope {
    static { /*static initializer; not a member */ }

    { /* instance initializer; not a member*/ }

    // static (class) field; a member
    public static int staticField = 1;

    // instance field; a member
    public int instanceField = 1;

    // static (class) method; a member
    public void aStaticMethod(){ }

    // instance method; a member
    public void anInstanceMethod(){}

    // constructor; not a member
    public MethodVariableScope(){ }
}
```

Not members

Members

Not a member

Members

From the Java Language Specification, §8.2:

Constructors, static initializers, and instance initializers are not members and therefore are not inherited.

Member Access Modifiers

Class Members	Same package	Same package or subclass	Same Module	Different Module
module alpha	*module alpha*	*module alpha*	*module alpha*	*module beta*
`package p1;`	`package p1;`	`package p2;`	`package p2;`	`package x;`
`class A {`	`class B {`	`class C extends A {`	`class D {`	`class E {`
`private int i;`				
`int j; // package-private`				
`protected int k;`				
`public int l;`				
`}`	`}`	`}`	`}`	`}`

Accessible **Inaccessible**

Members

Static and instance fields and methods

```
public class MethodVariableScope {

  public static int staticField = 1; // static field
  public int instanceVariable = 1;  // instance field

  public static void aStaticMethod(){
  }

  private void anInstanceMethod(){
  }
}
```

Members

Static and instance fields and methods

```
public class MethodVariableScope {

  public static int staticField = 1; // static field
  public int instanceVariable = 1;  // instance field

  public static void aStaticMethod(){
  }

  private void anInstanceMethod(){
  }
}
```

Members

Static and instance fields and methods

```
public class MethodVariableScope {

  public static int staticField = 1; // static field
  public int instanceVariable = 1;  // instance field

  public static void aStaticMethod(){
  }

  private void anInstanceMethod(){
  }
}
```

Members

Static and instance fields and methods

```
public class MethodVariableScope {

  public static int staticField = 1; // static field
  public int instanceVariable = 1;  // instance field

  public static void aStaticMethod(){
  }

  private void anInstanceMethod(){
  }
}
```

Members

Static and instance fields and methods

```
public class MethodVariableScope {

  public static int staticField = 1; // static field
  public int instanceVariable = 1;  // instance field

  public static void aStaticMethod(){
  }

  private void anInstanceMethod(){
  }
}
```

Method Variable Scope

Method Variable Scope

```
public class MethodVariableScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
 }
```

Method Variable Scope

Only code in the same scope can access a local variable

Method Variable Scope

```
public class MethodVariableScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
   }
```

Method Variable Scope

Only code in the same scope can access a local variable

Only code that follows the declaration of a local variable has access to that local variable

Method Variable Scope

```
public class MethodVaraibleScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
 }
```

Method Variable Scope

```
public class MethodVaraibleScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
 }
```

Method Variable Scope

```
public class MethodVaraibleScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
   }
```

Method Variable Scope

```
public class MethodVaraibleScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
   }
```

Method Variable Scope

```
public class MethodVaraibleScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
   }
```

Method Variable Scope

```
public class MethodVaraibleScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
   }
```

Method Variable Scope

```
public class MethodVaraibleScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
   }
```

Method Variable Scope

```
public class MethodVaraibleScope {
   public static void someMethod(int param1, int param2){
     int localVar0 = 0;

     if(true){
         int localVar1 = 0;
         if(true){
             int localVar2 = localVar1;
         }
     }else{
         int localVar1 = 2;
         for(int i = 0; i < 10; i++){
             int localVar2 = localVar1;
             // more code goes here
         }
     }
     int localVar3 = 3;
     while(true){
         int localVar4 = param2;
         // more code goes here
     }
   }
```

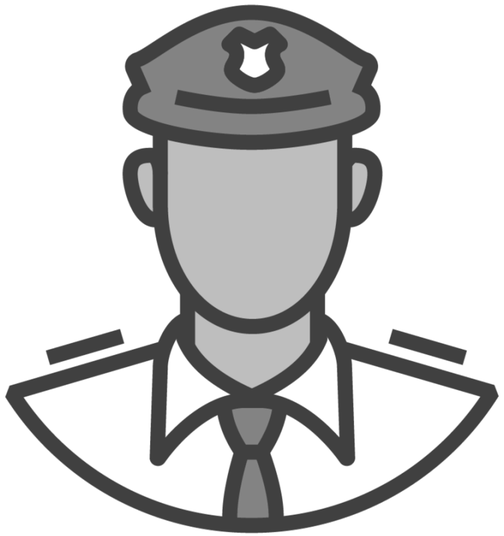
Method Variable Scope

Shadowing Local Variables

```
public class ShadowingAndScope {

   public static int memberVariable = 200;

    public void someMethod(){

      out.println(memberVariable); // 200


      out.println(memberVariable); // 2

      out.println(ShadowingAndScope.memberVariable);

    }
}
```

Variable Naming Rules

Naming Variables



Naming Rules

Must be followed or code will not compile

Naming Conventions

Are not required but are best practices

Variable Naming Rules

Variable name can be any length up to 65k

Use alphanumeric characters, dollar signs, or underscores

- **A→ Z**
- **a → z**
- **0 → 9**
- **_**
- **$**

Variable names are case sensitive

The first character must not be a number

No reserved words

- **Unless at least one character is not lower case.**

Variable Naming Rules

Allowed Characters (*A-Z, a-z, 0-9, $, _*)

```
int thisI$variaBLEname_thatIS5verylong = 0;

int ___$__$___  = 0;

int thisIsAlso-AVaraibleName = 0;

int valid_Variable = 0;
```

Variable Naming Rules

Case Sensitive

```
int someVArIABle = 0;
if( someVArIABle == 0){
    someVariable = 2;
}
```

Variable Naming Rules

Starting Character *(A-Z, a-z, $, _)*

```
int someMoney = 0;
int $omeMoney = 0;
int _moreMoney = 0;
int 1stMoney = 0;
```

Variable Naming Rules

List of Reserved Words

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

Variable Naming Rules

List of Reserved Words

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

Variable Naming Rules

No Reserved Words *(int, static, void, etc.)*

```
int theFinal = 0;
int the_final = 0;
int finalExam = 0;
int Final = 0;
int final = 0:
```

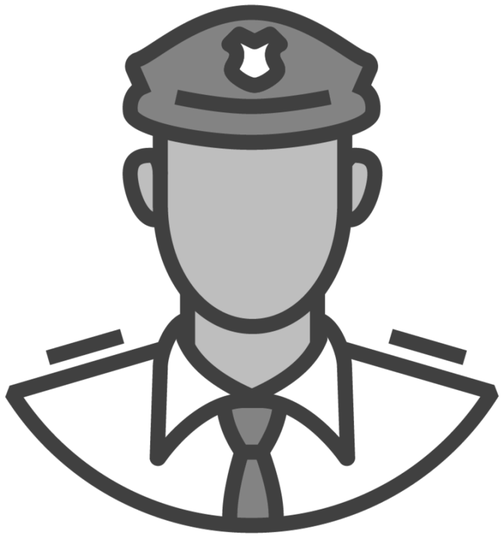
Variable Naming Rules

List of Reserved Words

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

Variable Naming Conventions

Naming Variables



Naming Rules

Must be followed or code will not compile

Naming Conventions

Are not required but are best practices

Variable Naming Conventions

- Avoid using $ and _
- Use whole words descriptive words
- Start with a lower-case letter
- A single word variable should be all lower case
- For names with multiple words use camel case

Variable Naming Conventions

```
int totalDue = 0;
int totalDue = 0;

int total = 0;

int total = 0;

int total = 0;

int totalAmountDue = 0;
```

Local Variable Type Inference

The var Variable Rules

Type is inferred

Initialized when declared

Type cannot change

Single variable declarations

Local variables only

The Varargs Parameter

The Varargs Paramter

m5.v6_Varargs

varargs

```
public class Varargs {

    public static void someMethod(int... nums){

      out.println(Arrays.toString(nums));





    }
}
```

>

The Varargs Paramter

m5.v6_Varargs

```
public class Varargs {

    public static void someMethod(int... nums){

      out.println(Arrays.toString(nums));

    }

    public static void main(String[] args) {

        someMethod(null);

    }
}
```

>

The Varargs Paramter

m5.v6_Varargs

```
public class Varargs {

    public static void someMethod(int... nums){

      out.println(Arrays.toString(nums));

    }

    public static void main(String[] args) {

        someMethod();

    }
}
```

>

The Varargs Paramter

m5.v6_Varargs

```
public class Varargs {

    public static void someMethod(int... nums){

      out.println(Arrays.toString(nums));

    }

    public static void main(String[] args) {

        someMethod(5);

    }
}
```

>

The Varargs Paramter

m5.v6_Varargs

```
public class Varargs {

    public static void someMethod(int... nums){

      out.println(Arrays.toString(nums));

    }

    public static void main(String[] args) {

        someMethod(3,5,7,9);

    }
}
```

>

The Varargs Paramter

m5.v6_Varargs

```
public class Varargs {

    public static void someMethod(int... nums){

      out.println(Arrays.toString(nums));

    }

    public static void main(String[] args) {

        int [] values = [2,4,6,8]

        someMethod(values);

    }
}
```

>

The Varargs Paramter

m5.v6_Varargs

```
public class Varargs {

    public static void someMethod( String x,
                                   double y,
                                   int... nums){

      out.print(x+", "); out.print(y + ", ");

      out.println(Arrays.toString(nums));
    }


    public static void main(String[] args) {

        someMethod("Hello",3.14,1,3,5);

    }
}
```

```
>
```

The Varargs Paramter

m5.v6_Varargs

```
public class Varargs {

    public static void someMethod(int... nums){
                                  String x,
                                  double y,

      out.print(x+", "); out.print(y + ", ");

      out.println(Arrays.toString(nums));
    }


    public static void main(String[] args) {

        someMethod(1,3,5,"Hello",3.14);

    }
}
```

> *Error*

The Varargs Paramter

m5.v6_Varargs

```
public class Varargs {

    public static void someMethod( String x,
                                   int... nums){
                                   double y,

      out.print(x+", "); out.print(y + ", ");

      out.println(Arrays.toString(nums));
    }


    public static void main(String[] args) {

        someMethod("Hello",1,3,5,3.14);

    }
}
```

> *Error*

Summary

- Members are fields and methods
- Reference types vs primitive types
- Fields initialized by default
- Local variables must be explicitly initialized
- Access modifiers
  - private, package-private (default), package, public
- Variable naming rules
- Variable naming conventions
- The var type
- The varargs Parameters

Up Next:
Working with Strings, Dates and Times