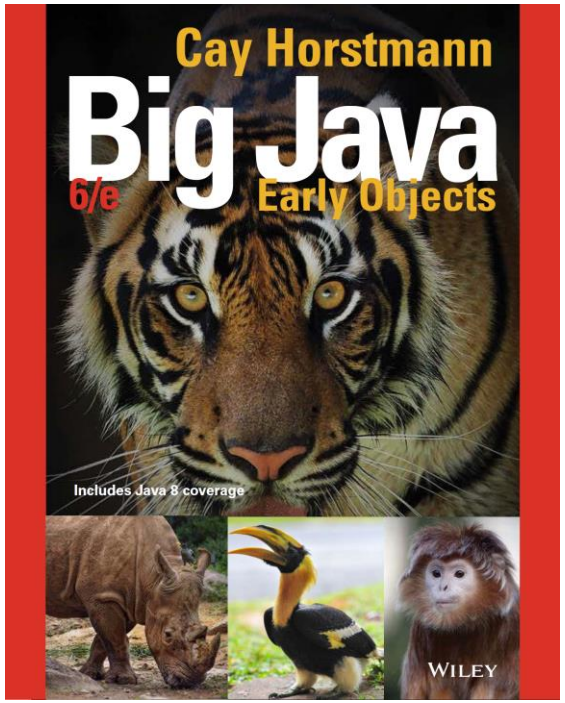


Big Java Early Objects by Cay Horstmann

Chapter 8 – Designing Classes Part II (Optional)

Preconditions

- **Precondition:** Requirement that the caller of a method must meet
- Publish preconditions so the caller won't call methods with bad parameters:

```
/**
   Deposits money into this account.
   @param amount the amount of money to deposit
   (Precondition: amount >= 0)
*/
```

- Typical use:
  1. *To restrict the parameters of a method*
  2. *To require that a method is only called when the object is in an appropriate state*

Preconditions

- If precondition is violated, method is not responsible for computing the correct result. It is free to do *anything*
- Method may throw exception if precondition violated — more in Chapter 11:

```
if (amount < 0) throw new IllegalArgumentException();
balance = balance + amount;
```

- Method doesn't have to test for precondition. (Test may be costly):

```
// if this makes the balance negative, it's the
// caller's fault
balance = balance + amount;
```

Preconditions

- Method can do an assertion check:

```
assert amount >= 0;
balance = balance + amount;
```

To enable assertion checking:

```
java -enableassertions MainClass
```

You can turn assertions off after you have tested your program, so that it runs at maximum speed

- Many beginning programmers silently return to the caller

```
if (amount < 0)
   return; // Not recommended; hard to debug
balance = balance + amount;
```

Syntax 8.1 Assertion

Syntax `assert` *condition*;

Example

```
assert amount >= 0;
```

If the condition is false and assertion checking is enabled, an exception occurs.

Condition that is claimed to be true.

Postconditions

- **Postcondition:** requirement that is true after a method has completed
- If method call is in accordance with preconditions, it must ensure that postconditions are valid
- There are two kinds of postconditions:
  - *The return value is computed correctly*
  - *The object is in a certain state after the method call is completed*

```
/**
   Deposits money into this account.
   (Postcondition: getBalance() >= 0)
   @param amount the amount of money to deposit
   (Precondition: amount >= 0)
*/
```

Postconditions

- Don’t document trivial postconditions that repeat the `@return` clause
- Formulate pre- and postconditions only in terms of the interface of the class:

```
amount <= getBalance() // this is the way to state a
   postcondition
amount <= balance // wrong postcondition formulation
```

- Contract: If caller fulfills preconditions, method must fulfill postconditions

Self Check 8.10

Why might you want to add a precondition to a method that you provide for other programmers?

Answer: Then you don't have to worry about checking for invalid values — it becomes the caller's responsibility.

Self Check 8.11

When you implement a method with a precondition and you notice that the caller did not fulfill the precondition, do you have to notify the caller?

Answer: No — you can take any action that is convenient for you.

Overlapping Scope

- A local variable can *shadow* a variable with the same name
- Local scope wins over class scope:

```
public class Coin
{
   ...
   public double getExchangeValue(double exchangeRate)
   {
      double value; // Local variable
      ...
      return value;
   }
   private String name;
   private double value; // variable with the same name
}
```

Overlapping Scope

- Access shadowed variables by qualifying them with the `this` reference:

```
value = this.value * exchangeRate;
```

Overlapping Scope

- Generally, shadowing an instance variable is poor code — error-prone, hard to read
- Exception: when implementing constructors or setter methods, it can be awkward to come up with different names for instance variables and parameters
- OK:

```
public Coin(double value, String name)
{
   this.value = value;
   this.name = name;
}
```

Self Check 8.16

Consider the following program that uses two variables named `r`.
Is this legal?

```
public class RectangleTester
{
   public static double area(Rectangle rect)
   {
      double r = rect.getWidth() * rect.getHeight();
      return r;
   }
   public static void main(String[] args)
   {
      Rectangle r = new Rectangle(5, 10, 20, 30);
      double a = area(r);
      System.out.println(r);
   }
}
```

Answer: Yes. The scopes are disjoint.

Self Check 8.17

What is the scope of the `balance` variable of the `BankAccount` class?

Answer: It starts at the beginning of the class and ends at the end of the class.

More Examples

```
class MyClass {

  private int myFirstObjectVariable = 3;

  private boolean mySecondObjectVariable = true;

  public void myFirstMethod() {

    int age = 12;

    // blah, blah, blah

    int greatAge = age * age;

    // blah, blah, blah

  }

  public void mySecondMethod() {

    boolean condition = true;

    int myNumber = 0;

    // blah, blah, blah (may change the value of condition)

    if (condition) {

      // blah, blah, blah

      int age = myNumber - 1;

      // blah, blah, blah

    }

    // blah, blah, blah

  }

}
```

two object variables - forget about them at the moment!

the scope of age

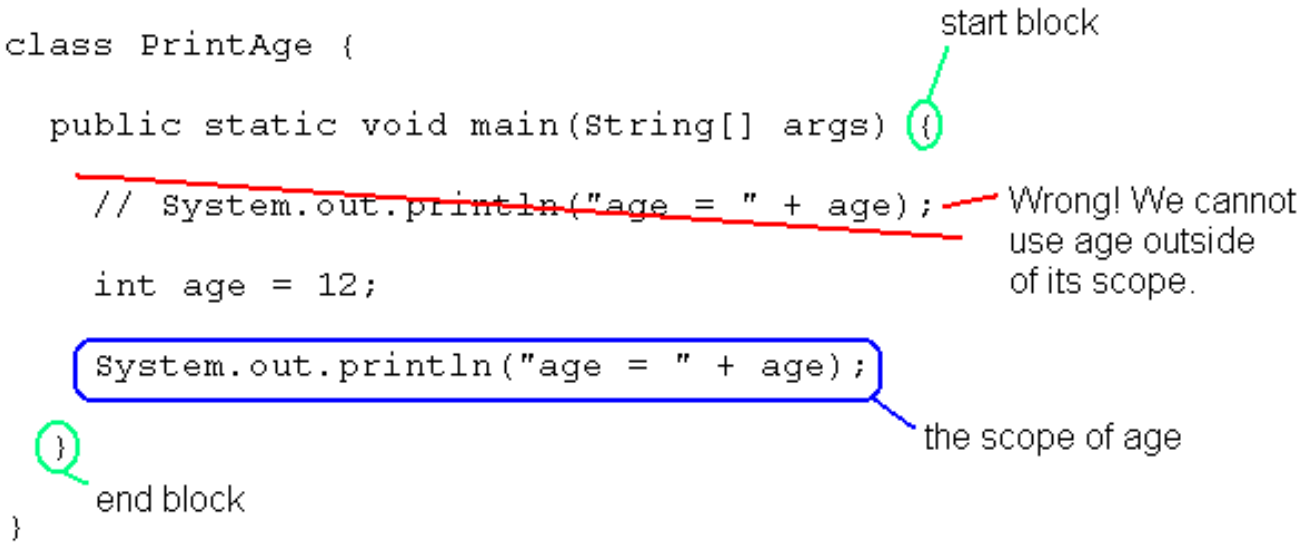
the scope of greatAge

the scope of condition

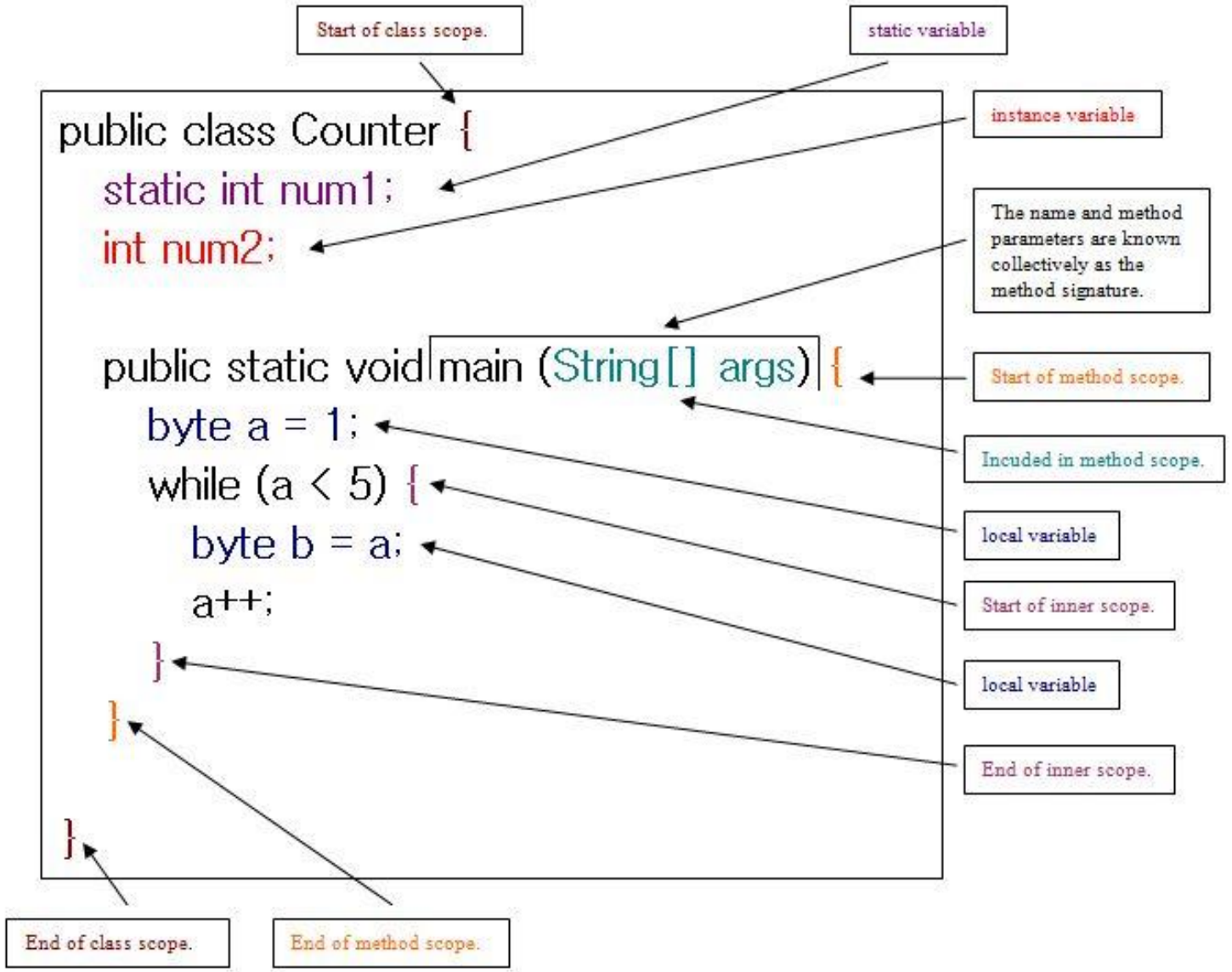
the scope of myNumber

the scope of age (it has nothing to do with the variable age in myFirstMethod())

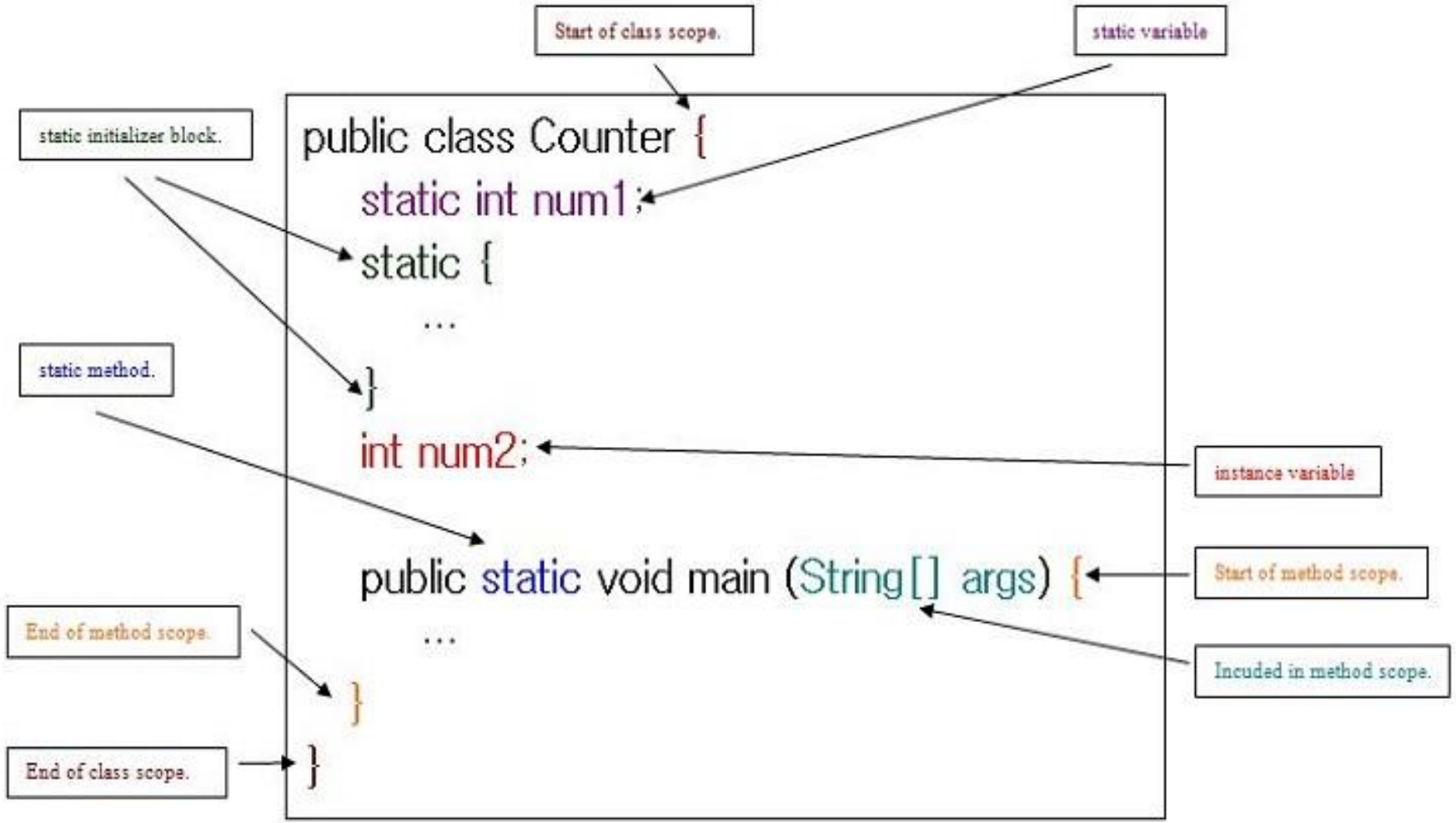
More Examples



More Examples



More Examples: Static Blocks



Packages

- **Package:** Set of related classes
- Important packages in the Java library:

Package	Purpose	Sample Class
`java.lang`	Language support	`Math`
`java.util`	Utilities	`Random`
`java.io`	Input and output	`PrintStream`
`java.awt`	Abstract Windowing Toolkit	`Color`
`java.applet`	Applets	`Applet`
`java.net`	Networking	`Socket`
`java.sql`	Database Access	`ResultSet`
`javax.swing`	Swing user interface	`JButton`
`omg.w3c.dom`	Document Object Model for XML documents	`Document`

Organizing Related Classes into Packages

- To put classes in a package, you must place a line

```
package packageName;
```

  as the first instruction in the source file containing the classes

- Package name consists of one or more identifiers separated by periods

Organizing Related Classes into Packages

- For example, to put the `Financial` class introduced into a package named `com.horstmann.bigjava`, the `Financial.java` file must start as follows:

```
package com.horstmann.bigjava;

public class Financial
{
   ...
}
```

- Default package has no name, no `package` statement

Syntax 8.2 Package Specification

Syntax `package` *packageName*;

Example

```
package com.horstmann.bigjava;
```

The classes in this file belong to this package.

A good choice for a package name is a domain name in reverse.

Importing Packages

- Can always use class without importing:

```
java.util.Scanner in = new java.util.Scanner(System.in);
```

- Tedious to use fully qualified name
- Import lets you use shorter class name:

```
import java.util.Scanner;
...
Scanner in = new Scanner(System.in)
```

- Can import all classes in a package:

```
import java.util.*;
```

- Never need to import `java.lang`
- You don’t need to import other classes in the same package

Package Names

- Use packages to avoid name clashes

  `java.util.Timer`

  vs.

  `javax.swing.Timer`

- Package names should be unambiguous
- Recommendation: start with reversed domain name:

  `com.horstmann.bigjava`

- `edu.sjsu.cs.walters`: for Britney Walters' classes (`walters@cs.sjsu.edu`)
- Path name should match package name:

  `com/horstmann/bigjava/Financial.java`

Package and Source Files

- **Base directory:** holds your program's Files
- Path name, relative to base directory, must match package name:

```
com/horstmann/bigjava/Financial.java
```

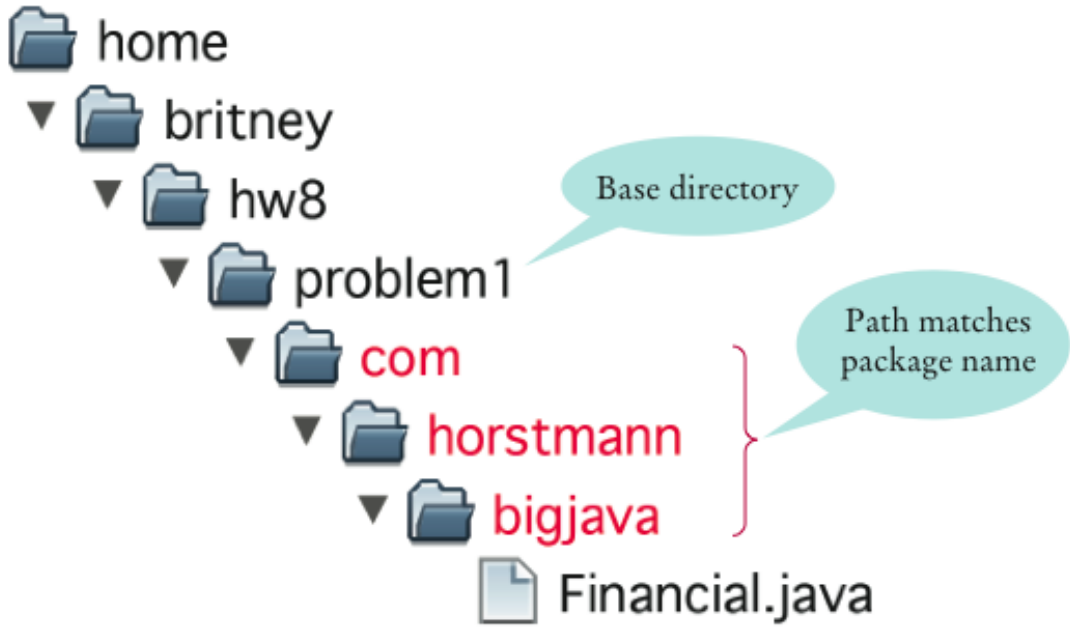


Figure 5
Base Directories and Subdirectories for Packages

Self Check 8.18

Which of the following are packages?

a. `java`

b. `java.lang`

c. `java.util`

d. `java.lang.Math`

Answer:

a.No

b.Yes

c.Yes

d.No

Self Check 8.19

Is a Java program without `import` statements limited to using the default and `java.lang` packages?

Answer: No — you simply use fully qualified names for all other classes, such as `java.util.Random` and `java.awt.Rectangle`.

Self Check 8.20

Suppose your homework assignments are located in the directory `/home/me/cs101` (`c:\Users\me\cs101` on Windows). Your instructor tells you to place your homework into packages. In which directory do you place the class `hw1.problem1.TicTacToeTester`?

Answer: `/home/me/cs101/hw1/problem1` or, on Windows, `c:\Users\me\cs101\hw1\problem1`