

# Object Oriented Programming - OOPs

By Watcharin Sarachai

# Topic

- Static Keyword
- Variable Scoping
- Package

# Static Keyword

# static Keyword in Java

- **static keyword** is mainly used for memory management.
- **static keyword** can be used with **variables**, **methods**, **blocks** and **nested classes**.
- **static keyword** is used to share the same variable or method of a given class.
- **static keyword** is used for a constant variable or a method that is same for every **instance of a class**.

# static Keyword in Java

- In Java programming language, **static** keyword is a non-access modifier and can be used for the following:
  - Static Block
  - Static Variable
  - Static Method
  - Static Classes

# Static Block

- If we need to do the computation in order to initialize your **static variables**, we can declare a **static block** that gets executed exactly once, when the class is first loaded.

```
// Java program to demonstrate the use of static blocks
public class BlockExample {
    // static variable
    static int j = 10;
    static int n;

    // static block
    static {
        System.out.println("Static block initialized.");
        n = j * 8;
    }

    public static void main(String[] args)
    {
        System.out.println("Inside main method");
        System.out.println("Value of j : "+j);
        System.out.println("Value of n : "+n);
    }
}
```

# Static Block

- When you execute the previous program, static block gets initialized and displays the values of the initialized variables.

## Output:

```
1 | Static block initialized
2 | Inside main method
3 | Value of j: 10
4 | Value of n : 80
```

# Static Block

- As you can see that both the static variables were initialized before we accessed them in the main method.

```
class StaticExample02 {  
    static int num;  
    static String mystr;  
    static {  
        num = 97;  
        mystr = "Static keyword in Java";  
    }  
  
    public static void main(String args[]) {  
        System.out.println("Value of num: " + num);  
        System.out.println("Value of mystr: " + mystr);  
    }  
}
```

Output:

Value of num: 97

Value of mystr: Static keyword in Java



# Example 2: Multiple Static blocks

- They execute in the given order which means the first static block executes before second static block.
- That's the reason, values initialized by first block are overwritten by second block.

```
class StaticExample03 {
    static int num;
    static String mystr;
    // First Static block
    static {
        System.out.println("Static Block 1");
        num = 68;
        mystr = "Block1";
    }
    // Second static block
    static {
        System.out.println("Static Block 2");
        num = 98;
        mystr = "Block2";
    }

    public static void main(String args[]) {
        System.out.println("Value of num: " + num);
        System.out.println("Value of mystr: " + mystr);
    }
}
```

Output:

```
Static Block 1
Static Block 2
Value of num: 98
Value of mystr: Block2
```

# Static Variable

- When we declare a variable as static, then a single copy of the variable is created and divided among all **objects** at the **class level**.
- Static variables are, essentially, global variables. Basically, all the instances of the class share the same static variable.
- Static variables can be created at class-level only.
- Static variables are also known as Class Variables.
- Unlike non-static variables, a static variables can be accessed directly in static and non-static methods.

# Example 1: Static variables can be accessed directly in Static method

- Here we have a static method `disp()` and two static variables `var1` and `var2`. Both the variables are accessed directly in the static method.

```
class StaticExample04 {  
    static int var1;  
    static String var2;  
  
    // This is a Static Method  
    static void disp() {  
        System.out.println("Var1 is: " + var1);  
        System.out.println("Var2 is: " + var2);  
    }  
  
    public static void main(String args[]) {  
        disp();  
    }  
}
```

Output:

```
Var1 is: 0  
Var2 is: null
```

# Static Variable

- Now let's understand this with the help of an example.

```
public class StaticExample05 {  
    // static variable  
    static int j = n();  
  
    // static block  
    static {  
        System.out.println("Inside the static block");  
    }  
  
    // static method  
    static int n() {  
        System.out.println("from n ");  
        return 20;  
    }  
  
    // static method(main !!)  
    public static void main(String[] args)  
    {  
        System.out.println("Value of j : "+j);  
        System.out.println("Inside main method");  
    }  
}
```

# Static Variable

- When you execute the above program, it will execute static block and the variable in order as defined in the above program.

## Output:

```
1 | from n
2 | Inside the static block
3 | Value of j: 20
4 | Inside main method
```

## Example 2: Static variables are shared among all the instances of class

- In this example, String variable is non-static and integer variable is Static.
- As you we see in the output that the non-static variable is different for both the objects but the static variable is shared among them, thats the reason the changes made to the static variable by object ob2 reflects in both the objects.

```

class StaticExample06 {
    // Static integer variable
    static int var1 = 77;
    // non-static string variable
    String var2;

    public static void main(String args[]) {
        StaticExample06 ob1 = new StaticExample06();
        StaticExample06 ob2 = new StaticExample06();
        /*
         * static variables can be accessed directly without any instances. Just to
         * demonstrate that static variables are shared, I am accessing them using
         * objects so that we can check that the changes made to static variables by one
         * object, reflects when we access them using other objects
         */
        // Assigning the value to static variable using object ob1
        ob1.var1 = 88;
        ob1.var2 = "I'm Object1";
        /*
         * This will overwrite the value of var1 because var1 has a single copy shared
         * among both the objects.
         */
        ob2.var1 = 99;
        ob2.var2 = "I'm Object2";
        System.out.println("ob1 integer:" + ob1.var1);
        System.out.println("ob1 String:" + ob1.var2);
        System.out.println("ob2 integer:" + ob2.var1);
        System.out.println("ob2 String:" + ob2.var2);
    }
}

```

Output:

```

ob1 integer:99
ob1 String:I'm Object1
ob2 integer:99
ob2 String:I'm Object2

```

# Static Methods

- When a **method** is declared with the *static* keyword, it is known as a *static method*.
- The most common example of a static method is the *main()* method.
- Methods declared as *static* can have the following restrictions:
  - They can directly call other static methods only.
  - They can access static data directly.
- **Syntax:**
  - Static keyword followed by return type, followed by method name.

```
static return_type method_name();
```



# Example 1: static method main is accessing static variables without object

```
class StaticExample07 {  
    static int i = 10;  
    static String s = "Beginnersbook";  
  
    // This is a static method  
    public static void main(String args[]) {  
        System.out.println("i:" + i);  
        System.out.println("s:" + s);  
    }  
}
```

Output:

```
i:10  
s:Beginnersbook
```

# Example 2: Static method accessed directly in static and non-static method

```
class StaticExample08 {
    static int i = 100;
    static String s = "Beginnersbook";

    // Static method
    static void display() {
        System.out.println("i:" + i);
        System.out.println("i:" + s);
    }

    // non-static method
    void funcn() {
        // Static method called in non-static method
        display();
    }

    // static method
    public static void main(String args[]) {
        StaticExample08 obj = new StaticExample08();
        // You need to have object to call this non-static method
        obj.funcn();

        // Static method called in another static method
        display();
    }
}
```

Output:

```
i:100
i:Beginnersbook
i:100
i:Beginnersbook
```

```

public class StaticExample09 {

    // static variable
    static int j = 100;

    // instance variable
    int n = 200;

    // static method
    static void a() {
        System.out.println("Print from a");

        // Cannot make a static reference to the non-static field b
        n = 100; // compilation error

        // Cannot make a static reference to the
        // non-static method a2() from the type Test
        a2(); // compilation error

        // Cannot use super in a static context
        System.out.println(super.j); // compiler error
    }

    // instance method
    void a2() {
        System.out.println("Inside a2");
    }

    public static void main(String[] args) {
        // main method
    }

}

```

In this examples, we can see how the restrictions are imposed on the **static methods**.

# Static Class

- A **class** can be made **static** only if it is a nested class.  
Nested static class doesn't need a reference of Outer class.
- In this case, a static class cannot access non-static members of the **Outer class**.
- A class can be made static only if it is a nested class.
  1. Nested static class doesn't need reference of Outer class
  2. A static class cannot access non-static members of the Outer class

# Static class Example

```
class StaticExample10 {  
    private static String str = "Hello, world!!";  
    // Static class  
    static class MyNestedClass {  
        // non-static method  
        public void disp() {  
  
            /*  
             * If you make the str variable of outer class non-static then you will get  
             * compilation error because: a nested static class cannot access non- static  
             * members of the outer class.  
             */  
            System.out.println(str);  
        }  
    }  
}  
  
public static void main(String args[]) {  
    /*  
     * To create instance of nested class we didn't need the outer class instance  
     * but for a regular nested class you would need to create an instance of outer  
     * class first  
     */  
    JavaExample.MyNestedClass obj = new JavaExample.MyNestedClass();  
    obj.disp();  
}
```

Output:

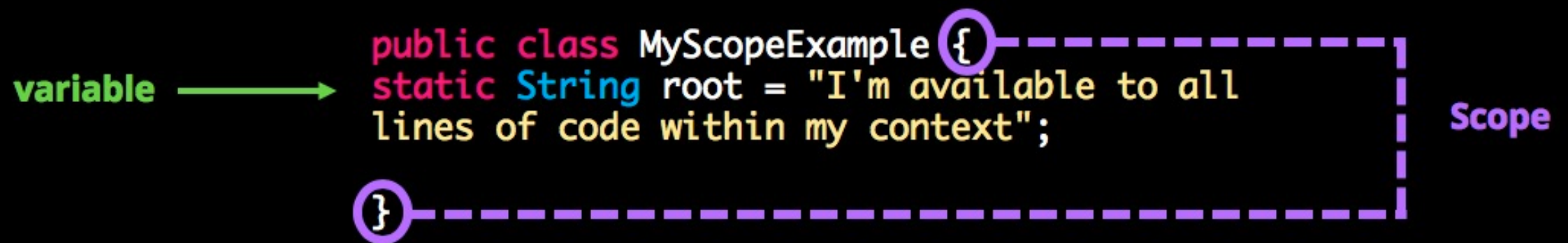
Hello, world!!

**Question ?**

# Variable Scoping

# Variable Scoping

- Each variable is only **available** (and accessible) **in the context within which it has been declared**.
- Most programming languages use curly braces (`{ }`) to mark the beginning and the end of a block of code.



The diagram shows a code snippet on a black background. The code is: `public class MyScopeExample { static String root = "I'm available to all lines of code within my context"; }`. The opening curly brace `{` is circled in purple. A green arrow points from the word **variable** to the `static String root` declaration. A dashed purple line connects the opening brace to the closing brace `}`, which is also circled in purple. To the right of the closing brace, the word **Scope** is written in purple.

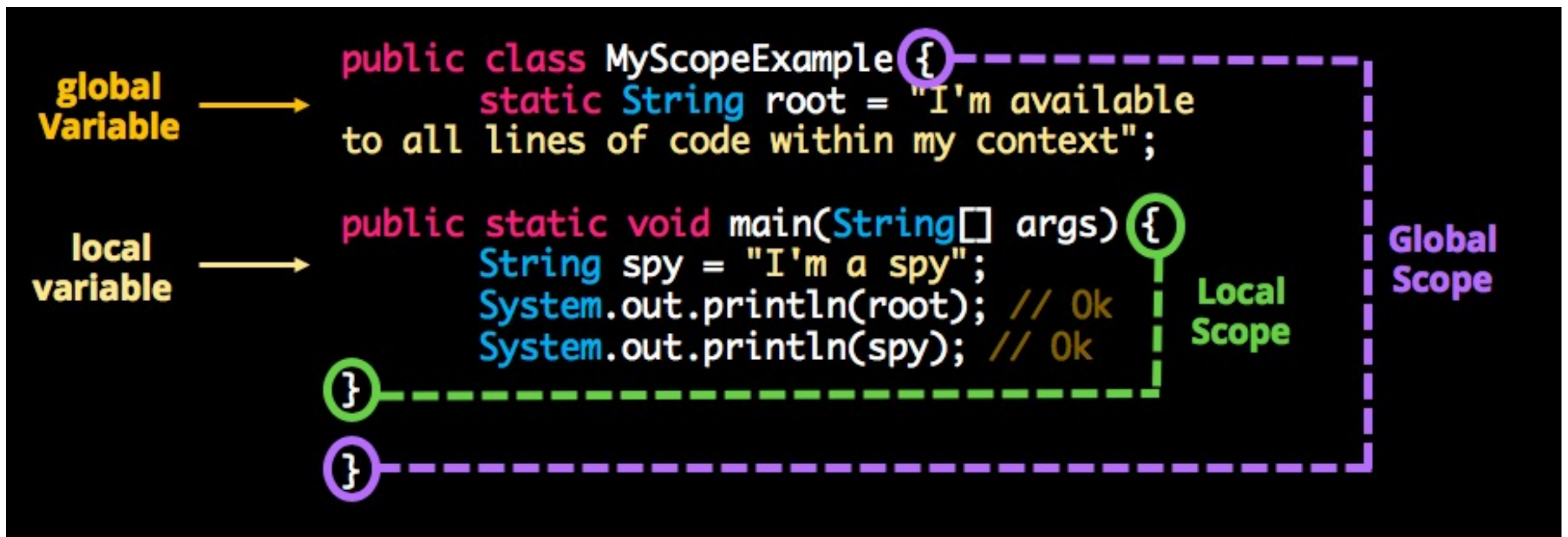
```
variable → public class MyScopeExample { static String root = "I'm available to all  
lines of code within my context"; }
```

Scope



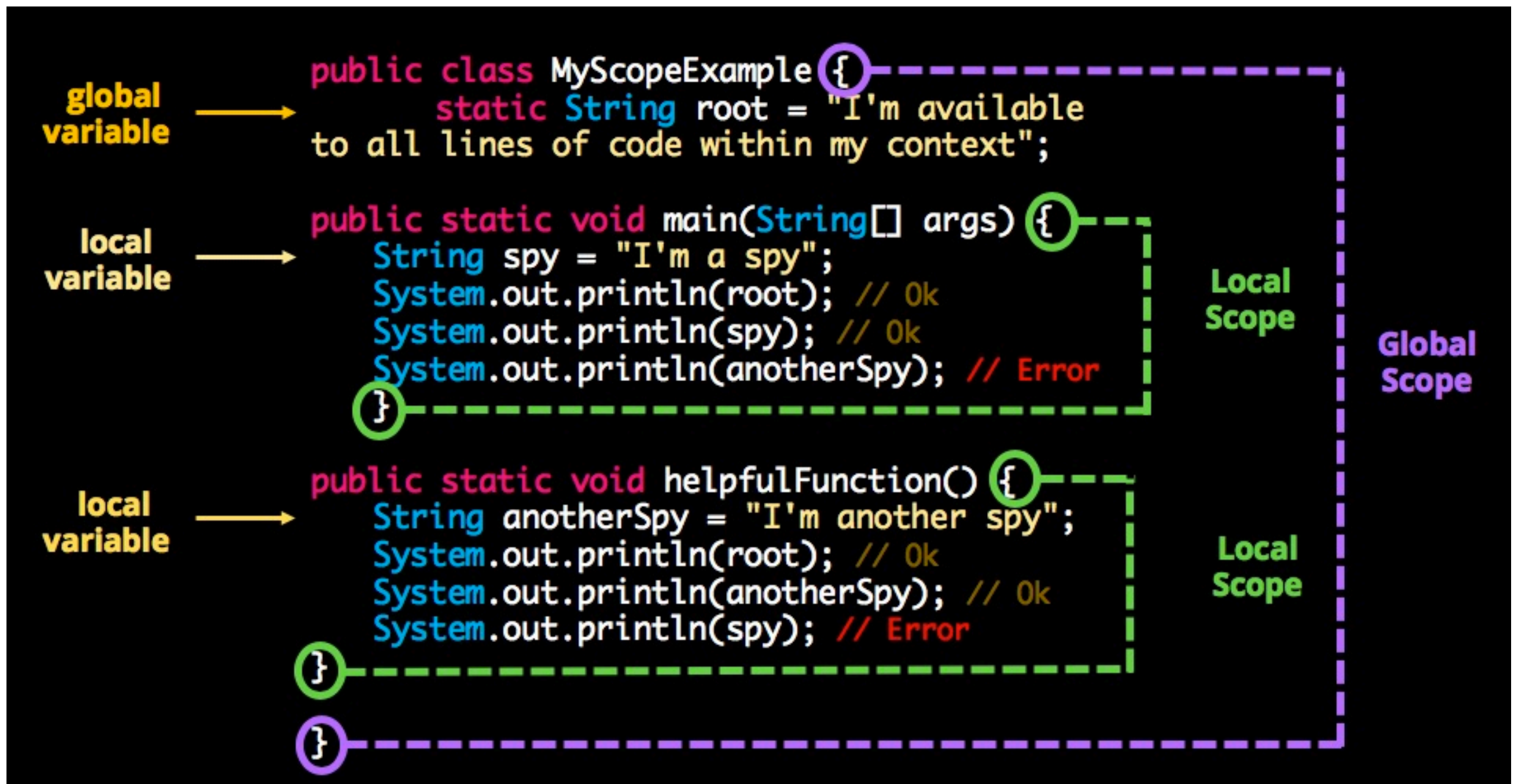
# Variable Scoping

- The scope of a variable can be **local** as well as **global**, depending on where a *variable is declared*.
- A global variable can be available to all the classes and methods within a program while a local variable might only be available within the method that it is declared in:



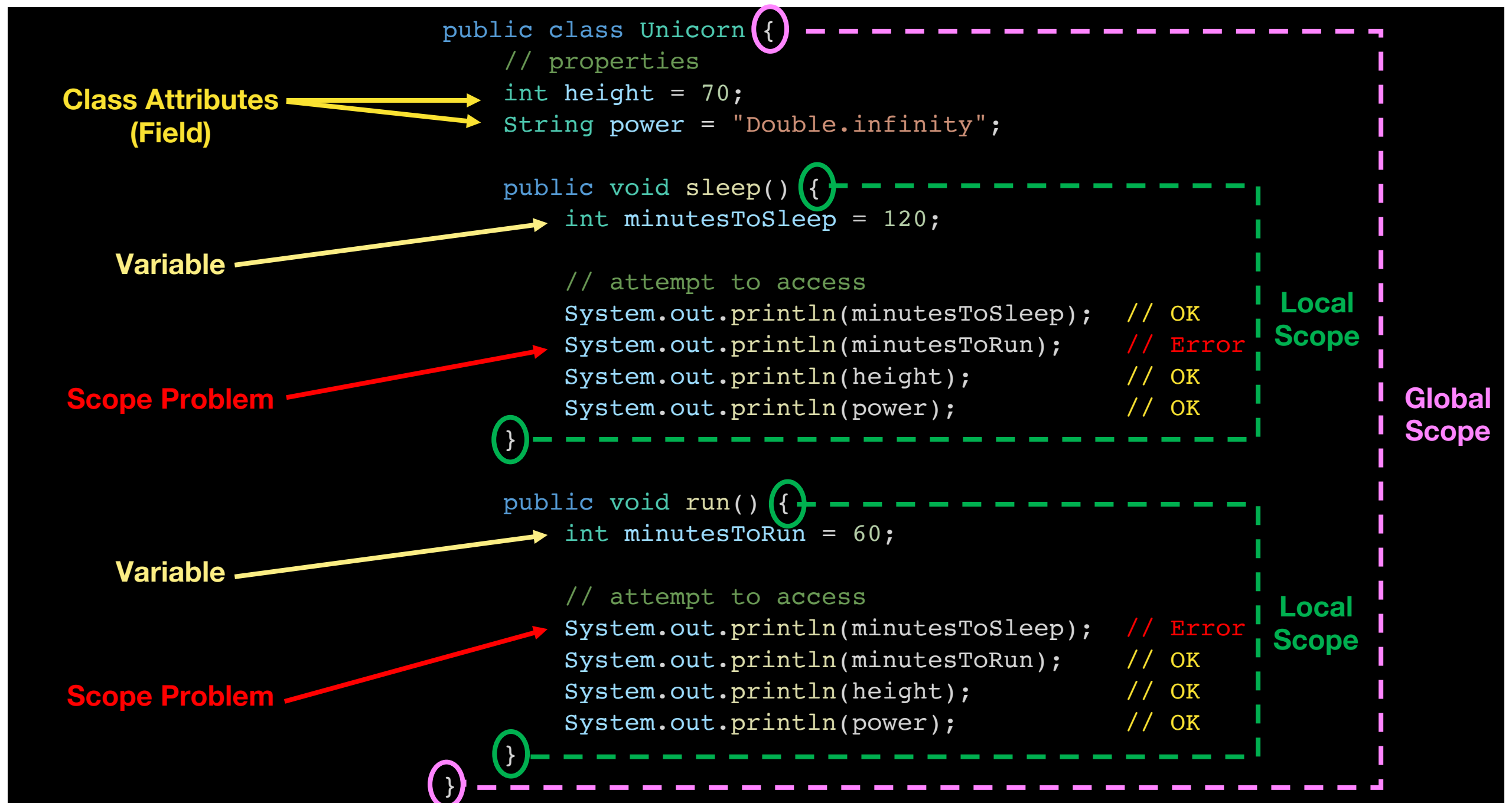
# Variable Scoping

- Whatever is in the local scope is not available to the global scope, or any other local blocks of code. Let's take a look at another example:



# Variable scope in classes

- When you declare a class, the same general rules of scoping apply: each variable is only accessible within its declaration block:



# Variable scope in classes

- There are **global** class variables as well as **local** ones. Let's review in detail:
  - The variables `height` and `power` are fields of the class and are accessible anywhere within the class.
  - The variable `minutesToSleep` is only accessible within the local scope of the block of code it's declared in.
  - The variable `minutesToRun` is only accessible within the local scope of the block of code it's declared in.

The scope of a variable limits its accessibility by definition. However, `class fields are accessible outside of the class` and can be used by any other block of code.

# Variable scope in classes

- In our example, those are `height` and `power`. If we declare a Unicorn variable, we can read or alter those values:

```
Unicorn firefly = new Unicorn();  
System.out.println("I know it's height: "+unicorn.height);  
// and can change its power!  
unicorn.power = 0;
```

# Try It Yourself

1. Run the code - what does the error message says after the javac command?
2. Remove the curly brackets around the insideCurlyBrackets variable
3. Run the code again



# Try It Yourself

Filename: MyScopes.java

```
public class MyScopes {  
    static String root = "I'm available to all lines  
of code within my context";  
  
    public static void main(String[] args) {  
        String spy = "I'm a spy";  
        //TODO Remove the curly brackets { and }  
        {  
            String insideCurlyBrackets="I'm an insider";  
        }  
        System.out.println(root);  
        System.out.println(spy);  
        System.out.println(insideCurlyBrackets);  
    }  
}
```

**Question ?**



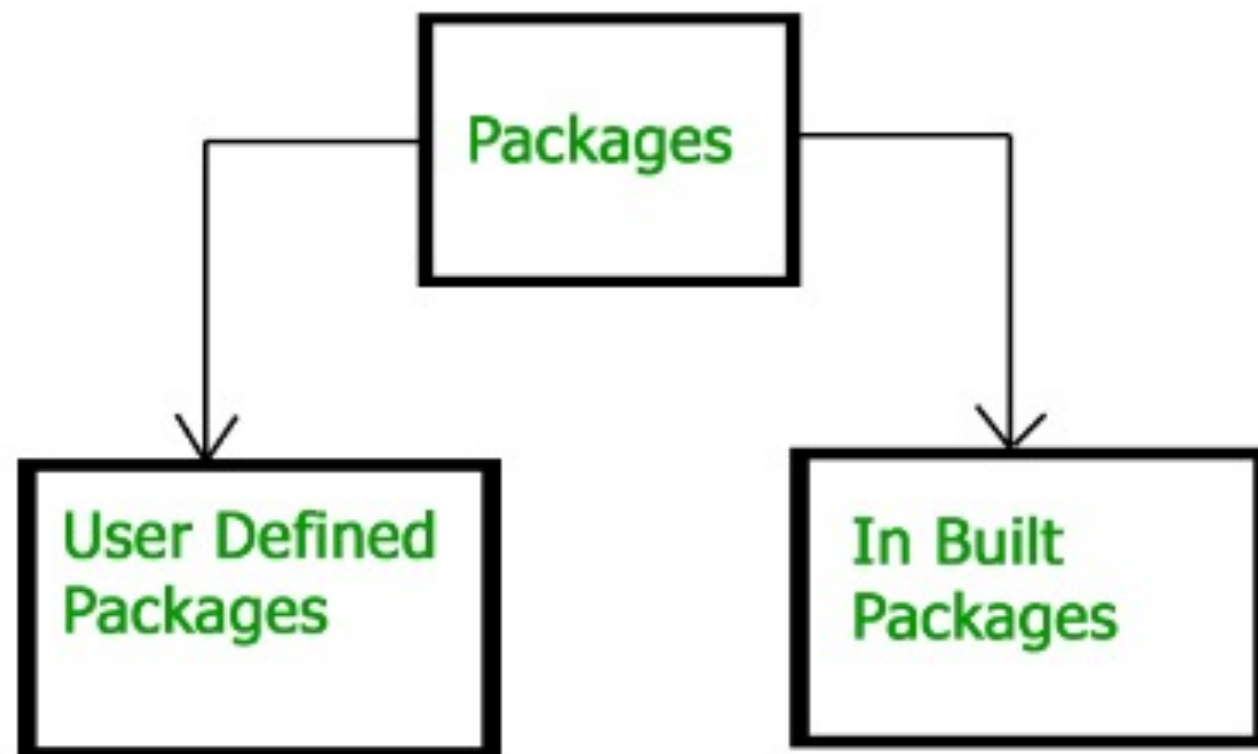
# Package

# Access Control

- A **package** in Java is used to group related classes together.
- Think of it as **a folder in a file directory**.
- We use packages to avoid name conflicts, and to write a better maintainable code.
- Putting all classes that does a similar tasks in place makes the development a lot easier.

# Access Control

- Packages are divided into two categories:
  - Built-in Packages (*packages from the Java API*)
  - User-defined Packages (*create your own packages*)



# Built-in Packages

- The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.
- The library contains components for **managing input**, **database programming**, and much more. The complete list can be found at Oracles website: <https://docs.oracle.com/javase/8/docs/api/>.
- The library is divided into **packages** and **classes**. Meaning you can either import a single class (*along with its methods and attributes*), or a whole package that contain all the classes that belong to the specified package.

# Built-in Packages

- To use a class or a package from the library, you need to use the **import** keyword:

## Syntax

```
import package.name.Class; // Import a single class
import package.name.*;     // Import the whole package
```

# Import Class

- If you want the Scanner class, **which is used to get user input**, write the following code:

## Example

```
import java.util.Scanner;
```

- In the example above, **java.util** is a package, while **Scanner** is a class of the **java.util** package.

# Import Class

- To use the **Scanner** class, create an object of the class and use any of the available methods found in the **Scanner** class documentation. In our example, we will use the **nextLine()** method, which is used to read a complete line:

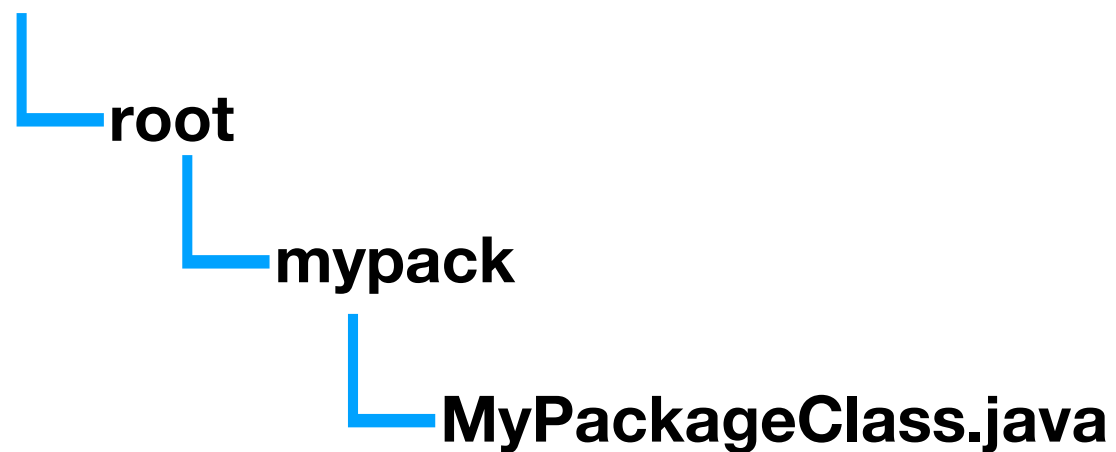
```
import java.util.*; // import the java.util package

public class MyClass {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);
        // Enter username and press Enter
        System.out.println("Enter username");
        String userName = myObj.nextLine();
        System.out.println("Username is: " + userName);
        myObj.close();
    }
}
```

# User-defined Packages

- To create your own **package**, you need to understand that Java uses a file system directory to store them. Just like **folders** on your computer:

## Example





# User-defined Packages

- To create a package, use the **package** keyword:

**MyPackageClass.java**

```
package camt.day1.packages;

class MyPackageClass {
    public static void main(String[] args) {
        System.out.println("This is my package!");
    }
}
```

# Try It Yourself

- Creates a package called **animals**. It is a good practice to use names of packages with lower case letters to avoid any conflicts with the names of classes and create the **Mammal** class:

## Mammal.java

```
package camt.day1.packages.animals;  
  
class Mammal {  
    public void eat() {  
        System.out.println("Mammal eats");  
    }  
    public void travel() {  
        System.out.println("Mammal travels");  
    }  
}
```

# Try It Yourself

- Creates a class called **RunTest** in the **camt** package and write the code as shown below:

**RunTest.java**

class RunTest is in  
camt package

Import class  
Mammal here

```
package camt;  
// TODO: import the class Mammal here  
import .....;  
  
class RunTest {  
    public static void main(String args[]) {  
        Mammal m = new Mammal();  
        m.eat();  
        m.travel();  
    }  
}
```

**Question ?**

# Lab 02

- Create a Java program to find a area of chape of circle, ragtangle, and triangle.
- Students must be designed the program to include at least four classes: Match, Circle, Rectangle, and Triangle classes.
- All mathematical calculations in the program must also be performed using Math class methods.
- Designed every class method to do its own behavior, for example, the "calArea()" method of Circle class must be calculated the area of a circle shape.

# Example code for get input from command-line prompt

```
package camt.ch02;

import java.util.Scanner;

public class CommandLineExample {
    public void test() {
        Scanner in = new Scanner(System.in);

        System.out.print("Please type a string: ");
        String s = in.nextLine();
        System.out.println("You entered string " + s);

        System.out.print("Please type a int value: ");
        int a = in.nextInt();
        System.out.println("You entered integer " + a);

        System.out.print("Please type a float value: ");
        float b = in.nextFloat();
        System.out.println("You entered float " + b);

        // closing scanner
        in.close();
    }
}
```

# Outout Example

```
1.Circle  
2.Rectangle  
3.Triangle  
Please select [1-3]: 1  
Enter the radius: 5  
The circle area is : 78.5
```

```
1.Circle  
2.Rectangle  
3.Triangle  
Please select [1-3]: 2  
Enter the width: 3  
Enter the height: 5  
The circle area is : 15.0
```

```
1.Circle  
2.Rectangle  
3.Triangle  
Please select [1-3]: 3  
Enter the base: 4  
Enter the height: 6  
The circle area is : 12.0
```

# Reference

- **Java Tutorial** <https://www.w3schools.com/java/> Accessed: 2020-07-04
- **Static Keyword in Java | Static Block, Variable, Method & Class | Edureka** <https://www.edureka.co/blog/static-keyword-in-java/> Accessed: 2020-07-04
- **Learn programming with Java – OpenClassrooms** <https://openclassrooms.com/en/courses/5667431-learn-programming-with-java> Accessed: 2020-07-04