While folks are joining

Get you laptops ready and login to your **replit** accounts.

We will be coding away in the session!

Crio Sprint : JAVA-2

Session 2

Today’s Session Agenda

- Java Access Modifiers
- Static keyword in Java
- Final keyword in Java
- Garbage Collection in Java

How would we do this with Java classes?

- We want some **variables to be accessible only to methods within the class** and not to methods outside the class - to ensure that we can validate values being assigned to these variables.
- We want to explicitly **specify that a method in the class can be used by anyone** since it provides the key functionality of the class.

Ans: **Access Specifiers**, let's get into more details.

Java Access Modifiers

- It is used to set the accessibility of **classes**, **constructors**, **methods**, and other **members** in Java.
- There are four access specifiers keyword in Java.
  - default (No keyword required)
  - **public**
  - **private**
  - **protected** (Will be discussed later), is slightly different from default

Default Access Modifier

- A **default** access modifier in Java has no specific keyword.
- **Whenever** the **access modifier is not specified**, then it is assumed to be the **default**.
- Default members are accessible only inside the **package**.
    - A package is a **namespace that organizes a set of related classes**.

```
class BaseClass
{
   void display()     //no access modifier indicates default modifier
     {
        System.out.println("BaseClass::display() with 'default' scope");
     }
}

class Main
{
   public static void main(String args[])
     {
        //access the class with default scope
        BaseClass obj = new BaseClass();

        obj.display();    //access class method with default scope
     }
}
```

Public Access Modifier

- A **public** access modifier is a modifier that does not restrict the members at all.
- A **public** member (method and fields) is accessible within the package as well as outside the package.

```
class A
{
    public void display()
    {
        System.out.println("Crio.Do!!");
    }
}
class Main
{
    public static void main(String args[])
    {
        A obj = new A ();
        obj.display();
    }

}
```

Curious Cats

- What's the disadvantage of making all fields and methods **public**?

Private Access Modifier

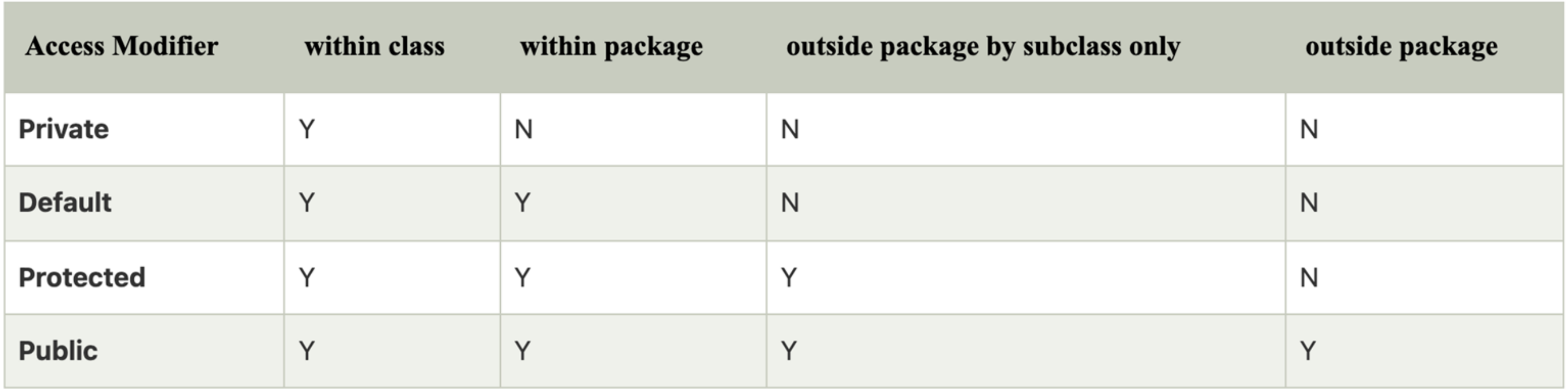
- The **private** access modifier is the one that has the lowest accessibility level.
- The scope of private entities (methods and fields) is **limited to the class in which they are declared**.
- Can you declare a constructor as private?
- A private access modifier **ensures encapsulation** in Java. (Will be discussed later.)

What will be the output?

```
class TestClass{
    //private variable and method
    private int num=100;
    private void printMessage(){
      System.out.println("Hello java");}
}

public class Main{
 public static void main(String args[]){
   TestClass obj=new TestClass();
   System.out.println(obj.num);
   obj.printMessage();
   }
}
```

Overall Summary of Access Specifiers



Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

How would we do this in Java?

- We want to **have a common variable across all instances of a class**, like the Company Name for an employee class.
  - This can be used to **keep count of number of instances created for a class**.
- We want to create a **method that is not related to a particular class instance but provides stand alone functionality**.
  - Example: Find the greater of two passed values.

Ans: **static keyword**, let’s get into more details.

Java static keyword

- It’s a member of a class **that isn’t associated with a specific instance** of the class.
- Can be **accessed without creating a new class instance**.
- A static member is **shared among all the instances** of the class.
- Used for
  - memory management
  - maintain information among all instances.
- Two important static members are:
  - **static variable/field**
  - **static method**

Static variable

- It's value is **common for all instances** of the class.
- It **gets memory only once** in the class area.
- Check Math.PI in the Math Java API and you'll find:
    - public static final double PI = 3.141592653589793;
    - Marked **public**, so accessible everywhere.
    - Marked **static**, so Math instance creation can be avoided.
    - Marked **final** (Will discuss it further).

What will be the output?

```
class Counter{
   static int count=0;//will get memory only
once and retain its value

   Counter(){
      count++;//incrementing the value of static
variable
      System.out.println(count);
   }

   public static void main(String args[]){
      //creating objects
      Counter c1=new Counter();
      Counter c2=new Counter();
      Counter c3=new Counter();
   }
}
```

Curious Cats

- When does memory for the static variable get allocated?
    - Static variables are initialized
        - when class is loaded.
        - before any object of that class is created.
        - before any static method of the class executes.

Java static method

- A static method means "behavior not dependent on an instance variable, so no instance/object is required. Just the class."
- Can be invoked without the need for any instance.
- Can access static member variable and modify it.
- Check Math Class in the Math Java API and you'll find:
  - Math.min(), Math.max() ,etc.

```
class Calculate{
    // static method
    static int cube(int x){
        return x*x*x;
    }

    public static void main(String args[]){
        int result=Calculate.cube(5);
       System.out.println(result);
    }
}
```

Things to know about Java static methods

What will be the output?

```
class Calculate{
  private int x = 3;
  static int cube(){
    return x*x*x;
  }
  public static void main(String args[]){
    int result = Calculate.cube();
    System.out.println(result);
  }
}
```

Static methods can’t use non-static (instance) variables.

What will be the output?

```
class Calculate{
  private int x = 3;
  public int getX(){
    return x;
  }
  static int cube(){
    return getX()*getX()*getX();
  }
  public static void main(String args[]){
    int result = Calculate.cube();
    System.out.println(result);
  }
}
```

Static methods can’t use non-static methods either!

Curious Cats

- Why is Java main method is static?
  - Stack Overflow Answer
- A static method can't access a non-static variable. But can a non-static method access a static variable?
  - Of course. A non-static method in a class can always call a static method in the class or access a static variable of the class.

Curious Cats

- Are static local variables (a variable with scope limited to function) allowed in Java?
  - Try executing the following code snippet.

```
class Main {
  public static void main(String args[]) {
    System.out.println(decrement());
  }

  static int decrement()
  {
    static int x= 10;
    return x--;
  }
}
```

- A static variable is a class variable (for whole class).
- Hence compiler does not allow static local variable.

How would we do this in Java?

- We want to define a constant value that no one can modify. E.g. double pi = 3.14;
- We want to create a class that cannot be inherited.
- We want to create a variable that references a particular object and no one can change it, to point to a different object.

Ans: **final** keyword, let’s get into more details.

final keyword in Java

- The final keyword is a non-access modifier used for classes, variables and methods, which **makes them non-changeable**.

What will be the output?

```
class Bike{
 final int speedlimit = 90; //final variable
 void run(){
    speedlimit = 400;
 }
 public static void main(String args[]){
    Bike obj=new  Bike();
    obj.run();

 }
```

- A **final variable** means **you can't change its value**.
- A **final method** means you can't override the method. (Will be discussed later)
- A **final class** means you can't extend the class. (Will be discussed later)

Blank variable in Java

- A **final variable that is not initialized** during declaration.
- **Initialized inside a constructor**.
- Useful at the **time of creating a object** and **once initialized may not be changed**.
    - For. eg. Aadhar Card number of an employee.

Guess the output?

```
public class Main {

    private final int blankFinalVariable;

    public static void main(String args[]) {
        Main clazz = new Main();
        System.out.println("Value of blank final 
variable is : " + clazz.blankFinalVariable);
    }


}
```

Failing to initialize will result in a compile-time error.

Static final variable in Java

- **static final** variables are used make variable constant.
- In other words, the value will remain unchanged as long as the class is loaded.
- Check Math.PI in the Math Java API and you'll find:
  - public static final double PI = 3.141592653589793;
  - Marked **public**, so accessible everywhere.
  - Marked **static**, so Math instance creation can be avoided.
  - Marked **final** because PI doesn't change.
- A good naming convention to consider:
  - Constant variable names should be in all caps!

Activity #1 - Which of the following would compile?

```
1. class Main {
  static int x;
  public void check(){
      System.out.println(x);
  }
  public static void main(String[] args)
{
    check();
  }
}
```

```
2. class Main {
  static final int x = 12;
  public void check(){
      System.out.println(x);
  }

  public static void main(String[]
args) {
    check();
  }
}
```

```
3. class Main {
  int x;
  public static void check(){
      System.out.println(x);
  }

  public static void main(String[]
args) {
    check();
  }
}
```

```
4. class Main {
  static final int x = 12;
  public void check(final int x){
    System.out.println(x);
  }

  public static void main(String[]
args) {
    check(10);
  }
}
```

```
5. class Main {

  final int x;
  public void check(){
      System.out.println(x);
  }

  public static void main(String[]
args) {
    check();
  }
}
```

```
6. class Main {

  int x = 12;
  public static void check(final int
x){
      System.out.println(x);
  }

  public static void main(String[]
args) {
    check(10);
  }
}
```

Garbage Collection in Java

- **Garbage Collection** is an **automatic process** of **reclaiming the runtime unused memory** by destroying the unused objects.
- Objects are created on the **heap**, which is a portion of memory dedicated to the program.
- At any point in time, the heap memory consists of two types of objects:
  - ***Live*** - these objects are being used and referenced from somewhere else
  - ***Dead*** - these objects are no longer used or referenced from anywhere
- The garbage collector finds these **Dead** objects and deletes them to free up memory.

How does Java know when to trigger Garbage Collection?

- Reference Out of Scope

```
void go () {
    Student s = new Student(); // reference 's' dies at end of method.
}
```

- By making a reference null

```
Student student = new Student();
student = null; // memory assigned by new can be reclaimed
```

- By re-assigning the reference variable to another variable

```
Student studentOne = new Student();
Student studentTwo = new Student();
studentOne = studentTwo; // now the first object referred by studentOne is available for garbage collection
```

- By using an anonymous object

```
register(new Student()); // The student object dies when function execution is complete.
```

Take home exercises for the session

- JAVA-2 Session 2 Quiz
- Takehome I - Replit

Feedback

Thank you for joining in today.

We’d love to hear your thoughts and feedback - https://docs.google.com/forms/d/e/1FAIpQLSer1fVLeYfU2rgPv0Br1UHnH-UQ9TocPGAxOIVZERIyQfziCg/viewform

Further Reading

- Access modifiers in Real life Usage
- Access Modifiers In Java - Tutorial With Examples (softwaretestinghelp.com)
- Java Program to implement private constructors (programiz.com)
- How to make object eligible for garbage collection in Java? - GeeksforGeeks
- Difference between static and instance member variables in Java? Answer | Java67
- Static keyword | Baeldung
- Final keyword | Baeldung

References

- Head First Java: A Brain-Friendly Guide, 2nd Edition (Covers Java 5.0) Book
- Learn Java Programming (programiz.com)
- Java Programming Language - GeeksforGeeks

Thank you