CS105 - Lab2

8 March 2022

INTERNATIONAL
UNIVERSITY OF SARAJEVO

Ş. Görkem Okur

Index

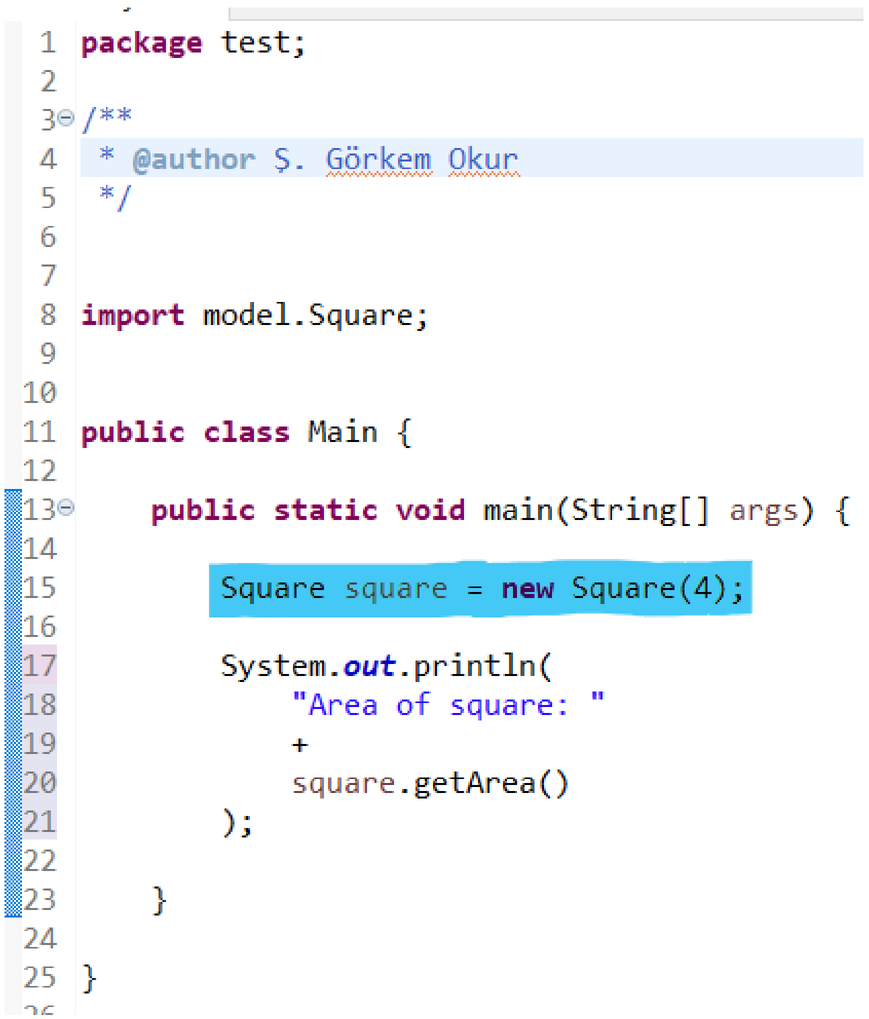
Class vs Primitive types

Class

A class is a blueprint or prototype from which objects are created (The Java™ Tutorials).

Object or Instance

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life (The Java™ Tutorials).



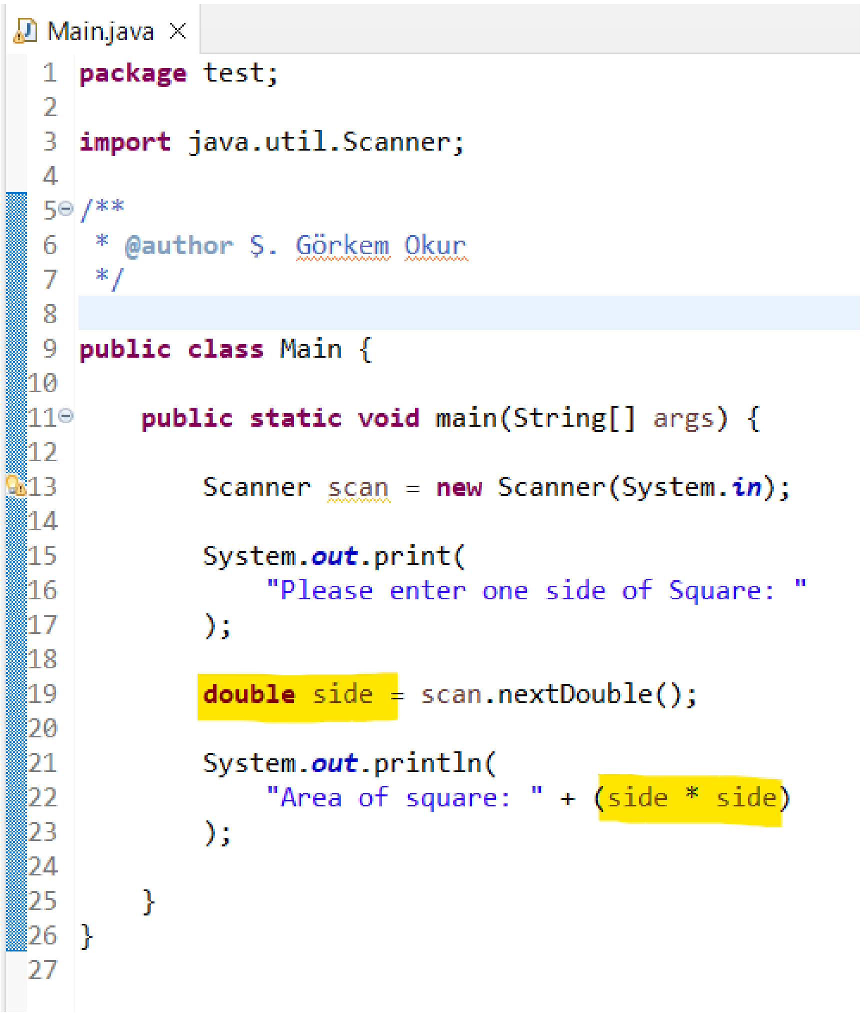
```
package test;

/**
 * @author Ş. Görkem Okur
 */


import model.Square;


public class Main {

    public static void main(String[] args) {

        Square square = new Square(4);

        System.out.println(
            "Area of square: "
            +
            square.getArea()
        );

    }

}
```

Class vs Primitive types

Primitive data types

A variable's data type determines the values it may contain, plus the operations that may be performed on it. the Java programming language supports eight primitive data types (The Java™ Tutorials).

Data types can be thought of as Containers. They keep the values, user given or calculated by program.


Main.java

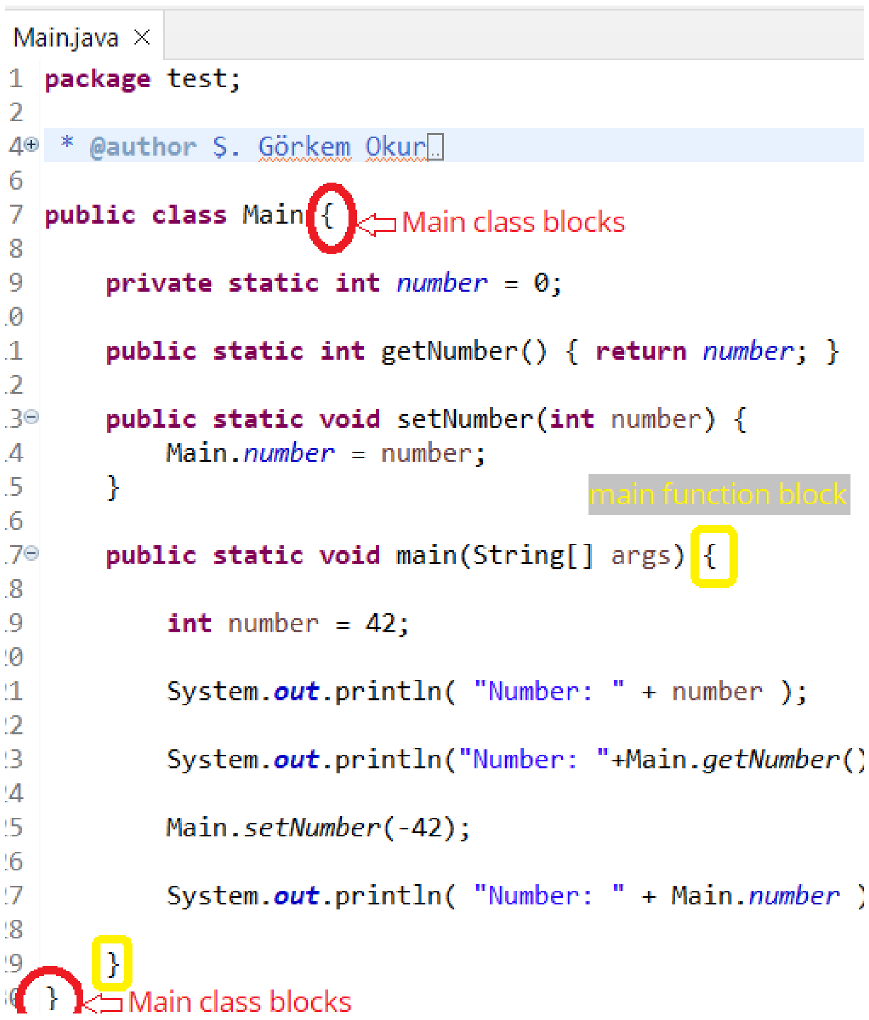
```java
package test;

import java.util.Scanner;

/**
 * @author Ş. Görkem Okur
 */

public class Main {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        System.out.print(
            "Please enter one side of Square: "
        );

        double side = scan.nextDouble();

        System.out.println(
            "Area of square: " + (side * side)
        );

    }
}
```

Blocks

- A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed (The Java™ Tutorials).

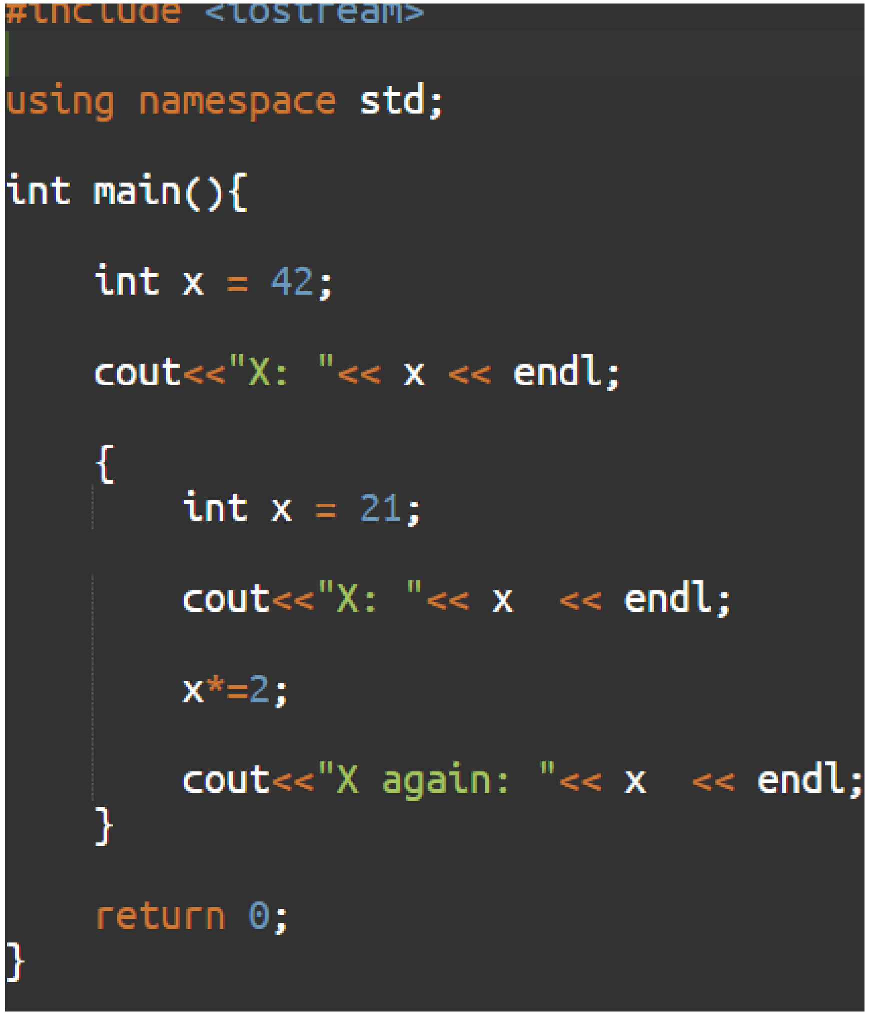
  We have three variables on this example and their name exactly same. However, compiler doesn't confuse which one is which.

- One is class variable, which is accessed by class name. To do this, we use dot(.) operator as access provider. It is used as access provide for objects and classes.

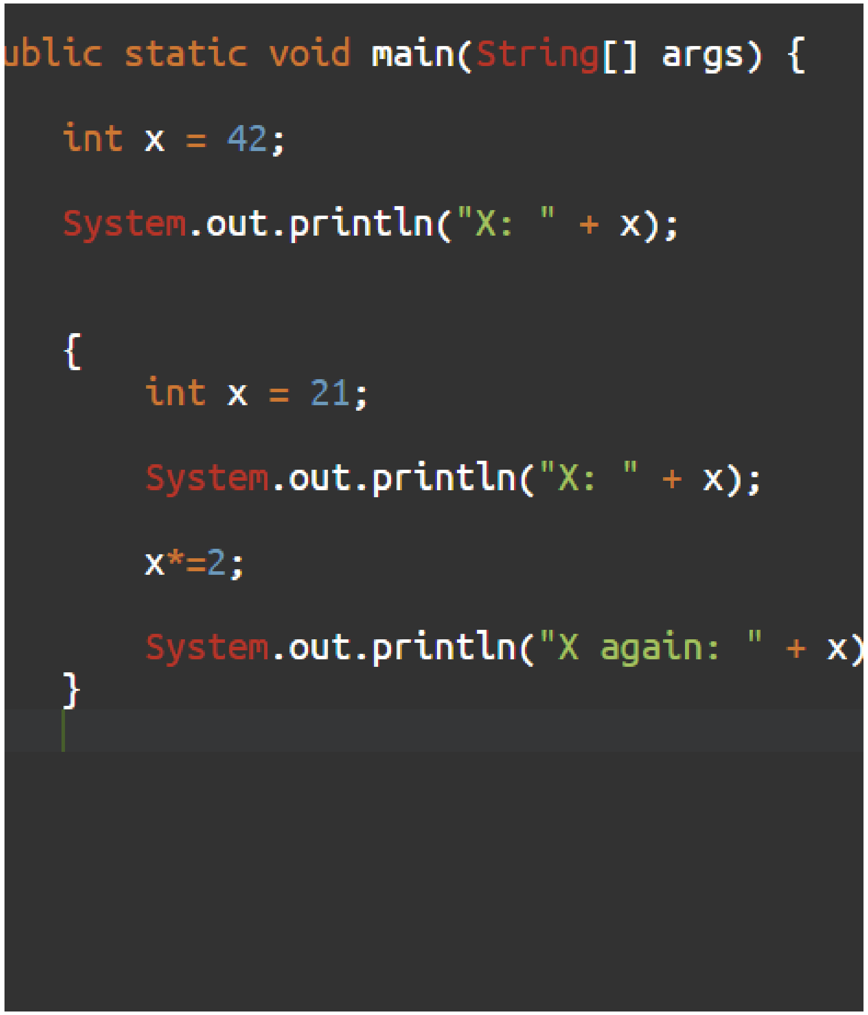


```java
package test;

 * @author Ş. Görkem Okur

public class Main {

    private static int number = 0;

    public static int getNumber() { return number; }

    public static void setNumber(int number) {
        Main.number = number;
    }

    public static void main(String[] args) {

        int number = 42;

        System.out.println( "Number: " + number );

        System.out.println("Number: "+Main.getNumber()

        Main.setNumber(-42);

        System.out.println( "Number: " + Main.number

    }
}
```

Blocks

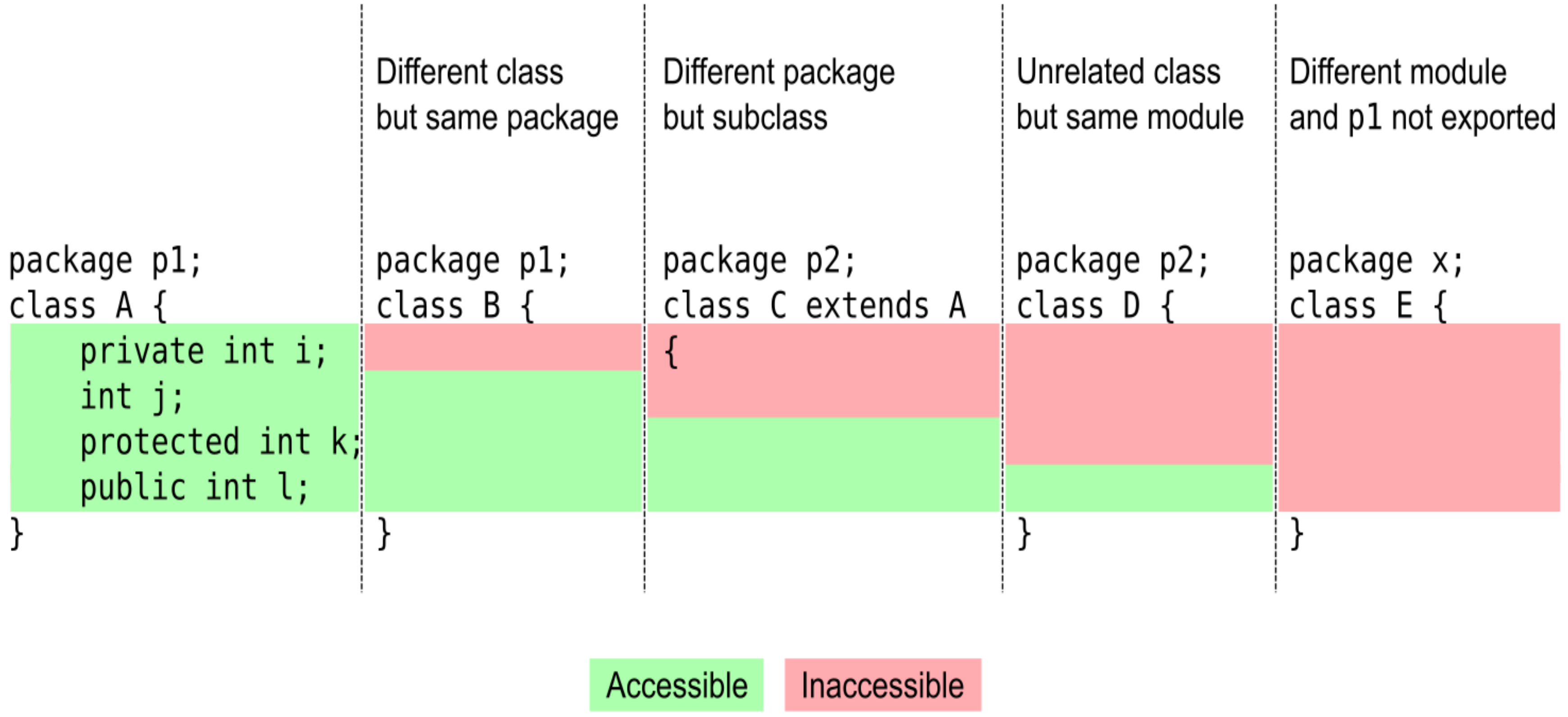


```
#include <iostream>

using namespace std;

int main(){

    int x = 42;

    cout<<"X: "<< x << endl;

    {
        int x = 21;

        cout<<"X: "<< x  << endl;

        x*=2;

        cout<<"X again: "<< x  << endl;
    }

    return 0;
}
```



```
ublic static void main(String[] args) {

    int x = 42;

    System.out.println("X: " + x);


    {
        int x = 21;

        System.out.println("X: " + x);

        x*=2;

        System.out.println("X again: " + x)
    }
```

Modifiers


Different class but same package

Different package but subclass

Unrelated class but same module

Different module and p1 not exported

```
package p1;
class A {
    private int i;
    int j;
    protected int k;
    public int l;
}
```

```
package p1;
class B {
}
```

```
package p2;
class C extends A
{
```

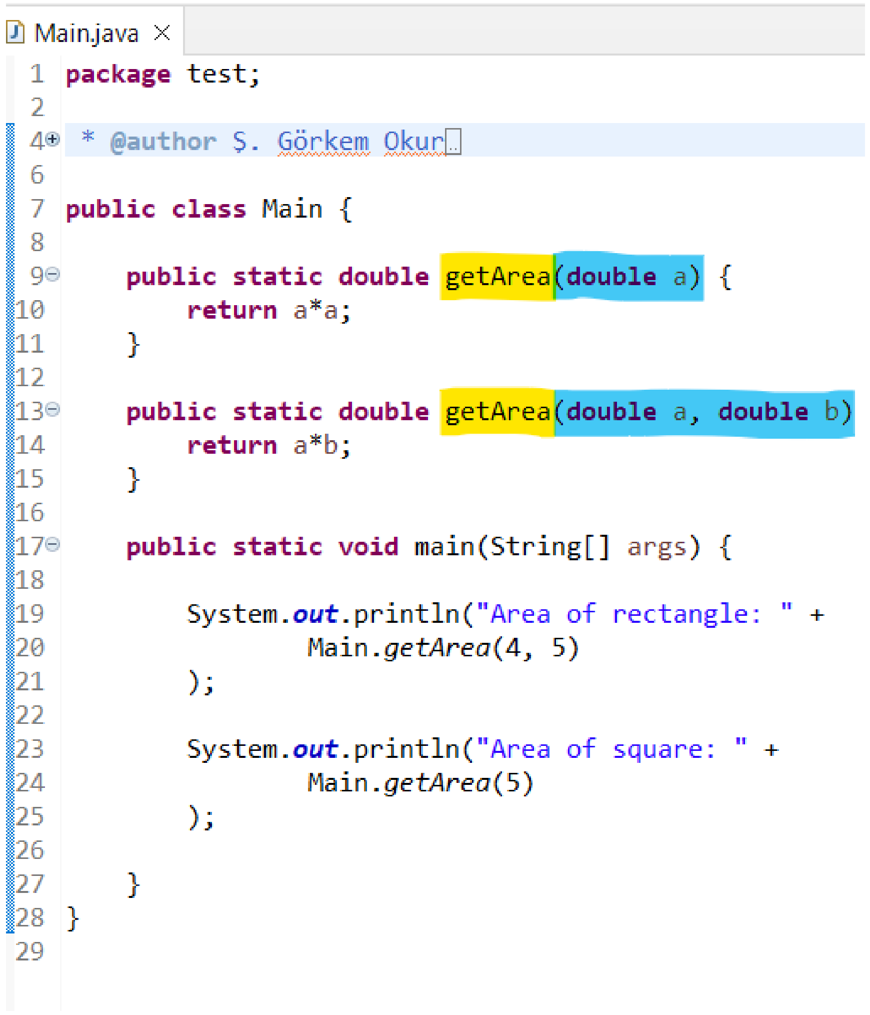
```
package p2;
class D {
}
```

```
package x;
class E {
}
```

🟩 Accessible 🟥 Inaccessible

Overloading

Java can distinguish between methods with different method signatures. This means that methods within a class can have the same name if they have different parameter lists (The Java™ Tutorials).



```
package test;

 * @author Ş. Görkem Okur

public class Main {

    public static double getArea(double a) {
        return a*a;
    }

    public static double getArea(double a, double b)
        return a*b;
    }

    public static void main(String[] args) {

        System.out.println("Area of rectangle: " +
                Main.getArea(4, 5)
        );

        System.out.println("Area of square: " +
                Main.getArea(5)
        );

    }
}
```

Overloading

```
public static int getArea(int a){
        return a*a;
}

public static float getArea(int a){
        return a*a;
}

public static void main(String[] args) {

System.out.println("Area of square: " + Main.getArea(5.0) );

}
```

Output: ???

Overloading

```
public static int getArea(int a){
        return a*a;
}

public static float getArea(int a){
        return a*a;
}

public static void main(String[] args) {

System.out.println("Area of square: " + Main.getArea((int)5.0) );

}
```

Output: ???

Overloading

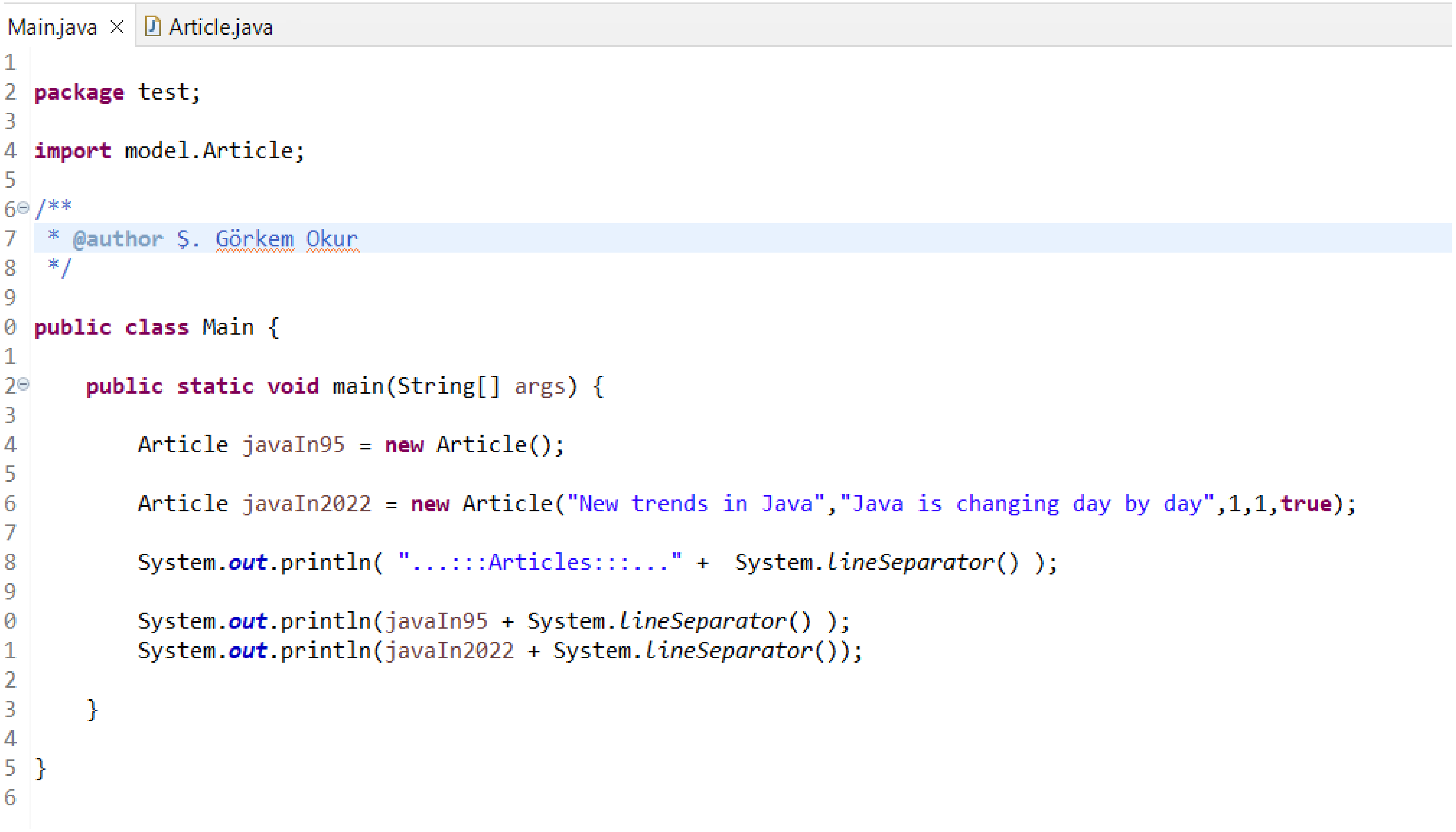
```
public static int getArea(int a){
        return a*a;
}
public static double getArea(double a){
        return a*a;
}
public static void main(String[] args) {
System.out.println("Area of square: " + Main.getArea(5) );
}
```

Output: ???

Constructors

- Constructors are invoked when the object is created first.
- To invoke a constructor, new keyword should be used.
- Constructors could be used just ones, that is the time of creation of object.
- After creation, constructors could not be invoked.

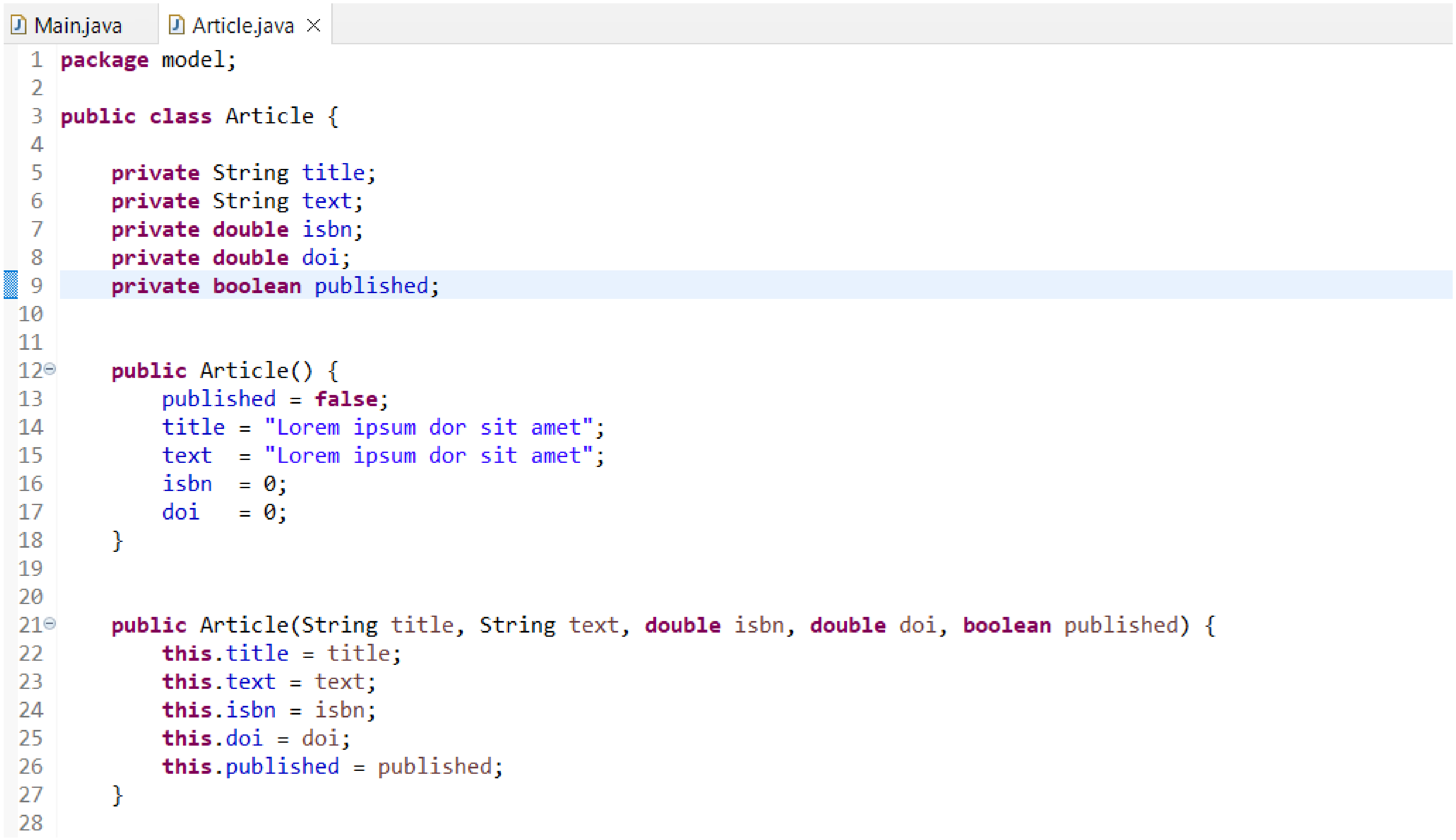
Constructors



Main.java × Article.java

```

package test;

import model.Article;

/**
 * @author Ş. Görkem Okur
 */

public class Main {

    public static void main(String[] args) {

        Article javaIn95 = new Article();

        Article javaIn2022 = new Article("New trends in Java","Java is changing day by day",1,1,true);

        System.out.println( "...:::Articles:::..." +  System.lineSeparator() );

        System.out.println(javaIn95 + System.lineSeparator() );
        System.out.println(javaIn2022 + System.lineSeparator());

    }

}

```

Constructors

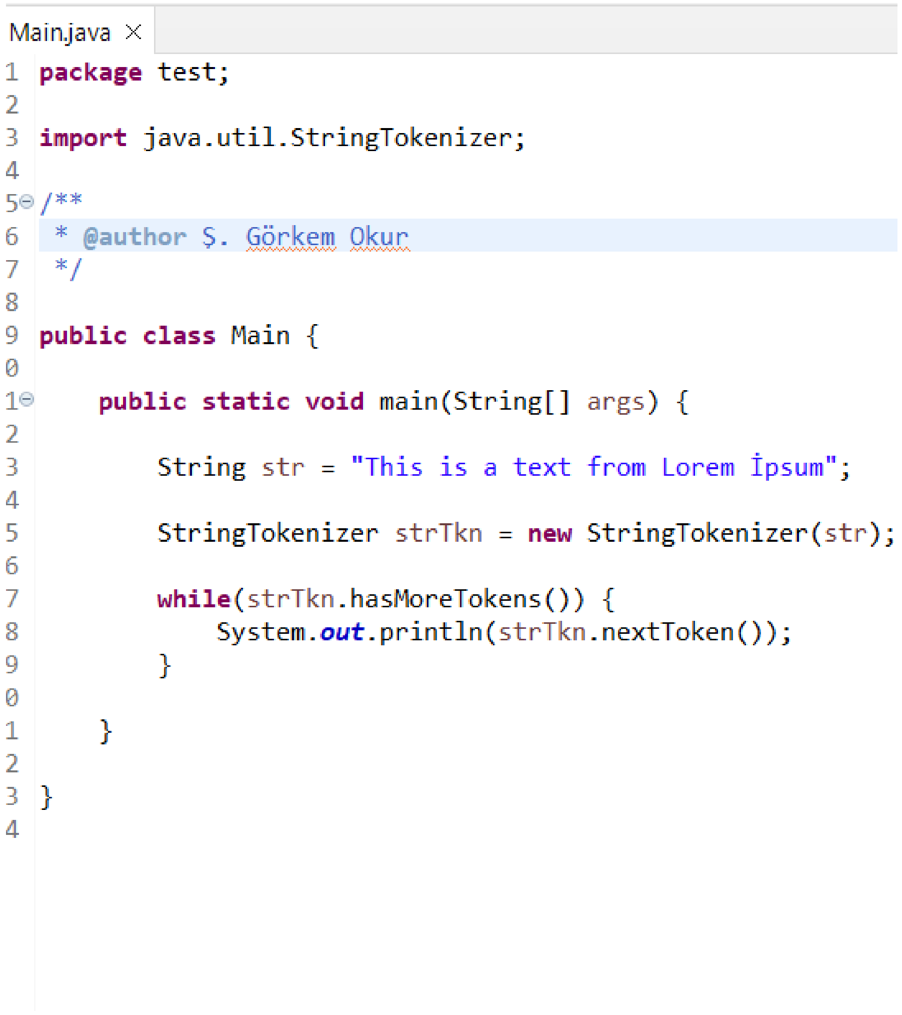


```java
package model;

public class Article {

    private String title;
    private String text;
    private double isbn;
    private double doi;
    private boolean published;


    public Article() {
        published = false;
        title = "Lorem ipsum dor sit amet";
        text  = "Lorem ipsum dor sit amet";
        isbn  = 0;
        doi   = 0;
    }


    public Article(String title, String text, double isbn, double doi, boolean published) {
        this.title = title;
        this.text = text;
        this.isbn = isbn;
        this.doi = doi;
        this.published = published;
    }

```

StringTokenizer

The string tokenizer class allows an application to break a string into tokens. The tokenization method is much simpler than the one used by the StreamTokenizer class. The StringTokenizer methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments. (The Java™ Tutorials).

- **Output:**
  - This
  - is
  - a
  - text
  - from
  - Lorem
  - İpsum



```java
package test;

import java.util.StringTokenizer;

/**
 * @author Ş. Görkem Okur
 */

public class Main {

	public static void main(String[] args) {

		String str = "This is a text from Lorem İpsum";

		StringTokenizer strTkn = new StringTokenizer(str);

		while(strTkn.hasMoreTokens()) {
			System.out.println(strTkn.nextToken());
		}

	}

}
```

Exercise

(Dice Rolling)

Write an application to simulate the rolling of two dice. The application should use an object of class Random once to roll the first die and again to roll the second die. The sum of the two values should then be calculated. Each die can show an integer value from 1 to 6, so the sum of the values will vary from 2 to 12, with 7 being the most frequent sum, and 2 and 12 the least frequent. Your application should roll the dice 36,000,000(Many) times. Use a one-dimensional array to tally the number of times each possible sum appears. Display the results in tabular format.

Exercise

(Airline Reservations System)

A small airline has just purchased a computer for its new automated reservations system. You've been asked to develop the new system. You're to write an application to assign seats on each flight of the airline's only plane (**capacity: 10 seats**). Your application should display the following alternatives**:**

Please type 1 for First Class and Please type 2 for Economy.

If the user types 1, your application should assign a seat in the **firstclass** section (**seats 1–5**).

If the user types 2, your application should assign a seat in the **economy** section (**seats 6–10**).

Your application should then display a boarding pass indicating the person's seat number and whether it's in the first-class or economy section of the plane.

Use a one-dimensional array of primitive type boolean to represent the seating chart of the plane. Initialize all the elements of the array to false to indicate that all the seats are empty. As each seat is assigned, set the corresponding element of the array to true to indicate that the seat is no longer available. **Your application should never assign a seat that has already been assigned.** When the economy section is full, your application should ask the person if it's acceptable to be placed in the first-class section (and vice versa). If yes, make the appropriate seat assignment. If no, display the message "Next flight leaves in 3 hours."

Thank You For Your Attention

Ş. Görkem Okur

References

1. https://docs.oracle.com/javase/tutorial/java/concepts/index.html
2. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
3. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html
4. https://i.stack.imgur.com/hzv3z.png
5. https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html
6. https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html