Strings

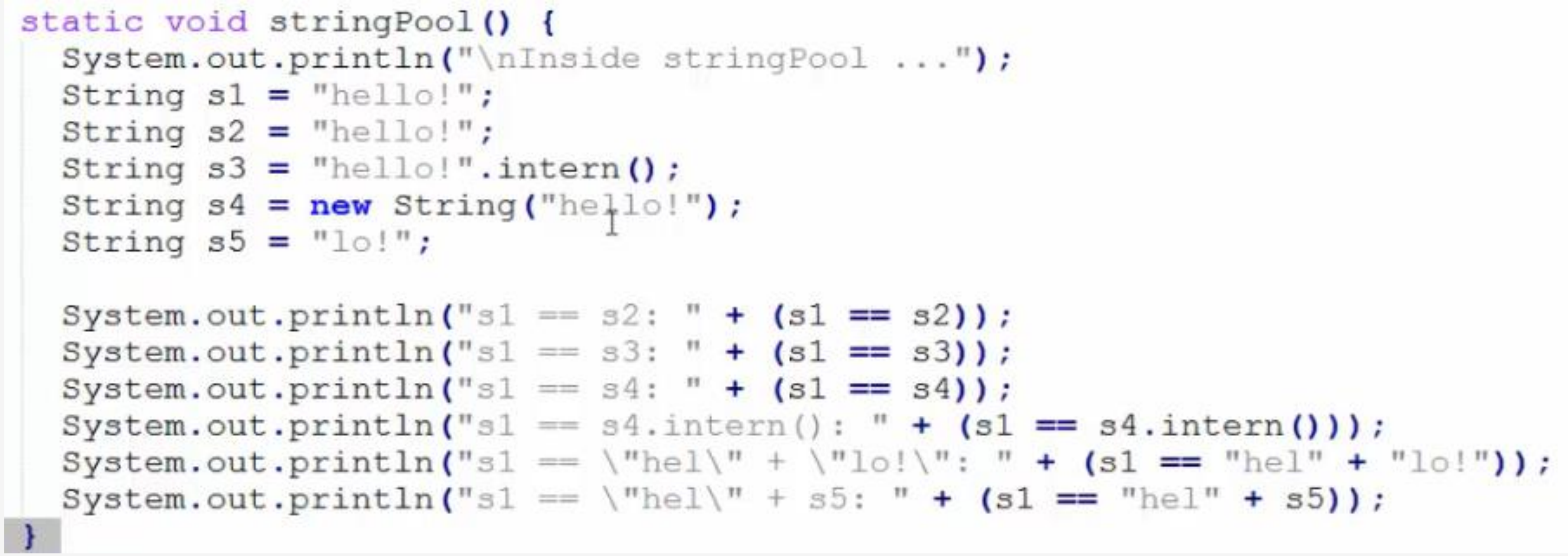
ФПМИ, „Информатика и Софтуерни Науки“ доц. д-р инж. М. Маринова

Класът String съдържа 50 метода. Често се налага създаването на стрингове от по-малки стрингове по време на програмирането. Не е удачно да се използва конкантенация на стринговете, защото всеки път когато правим конкатенация се създава нов обект от тип String. Така се използва ненужно паметта. Това се разрешава с ползването на StringBuilder класа.

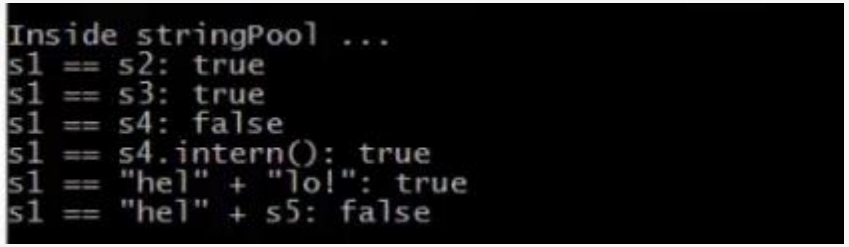
```
String s = “hel” + “lo”; ~intern

String s1 = “lo”;

String s2 = “hel” + s1; ~not interned


s2 = s2.intern(); ~explicit interning
```

STRING INTERN

```
String s = “hel” + “lo”; ~intern

String s1 = “lo”;

String s2 = “hel” + s1; ~not interned


s2 = s2.intern(); ~explicit interning
```



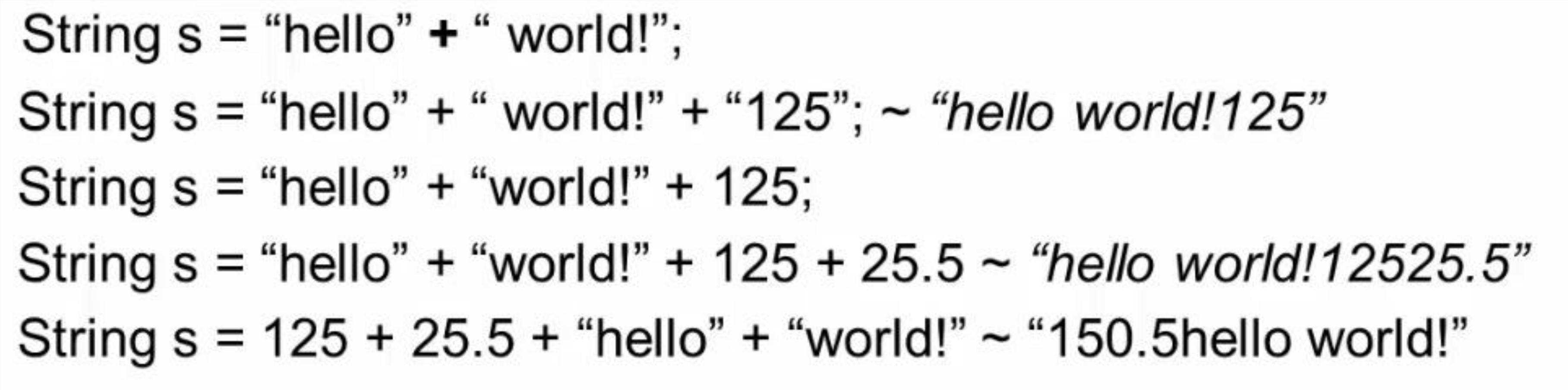
```
static void stringPool() {
    System.out.println("\nInside stringPool ...");
    String s1 = "hello!";
    String s2 = "hello!";
    String s3 = "hello!".intern();
    String s4 = new String("hello!");
    String s5 = "lo!";

    System.out.println("s1 == s2: " + (s1 == s2));
    System.out.println("s1 == s3: " + (s1 == s3));
    System.out.println("s1 == s4: " + (s1 == s4));
    System.out.println("s1 == s4.intern(): " + (s1 == s4.intern()));
    System.out.println("s1 == \"hel\" + \"lo!\": " + (s1 == "hel" + "lo!"));
    System.out.println("s1 == \"hel\" + s5: " + (s1 == "hel" + s5));
}
```



```
Inside stringPool ...
s1 == s2: true
s1 == s3: true
s1 == s4: false
s1 == s4.intern(): true
s1 == "hel" + "lo!": true
s1 == "hel" + s5: false
```

Събиране на стрингове

- **Използва се оператора +**



```
String s = “hello” + “ world!”;
String s = “hello” + “ world!” + “125”; ~ “hello world!125”
String s = “hello” + “world!” + 125;
String s = “hello” + “world!” + 125 + 25.5 ~ “hello world!12525.5”
String s = 125 + 25.5 + “hello” + “world!” ~ “150.5hello world!”
```

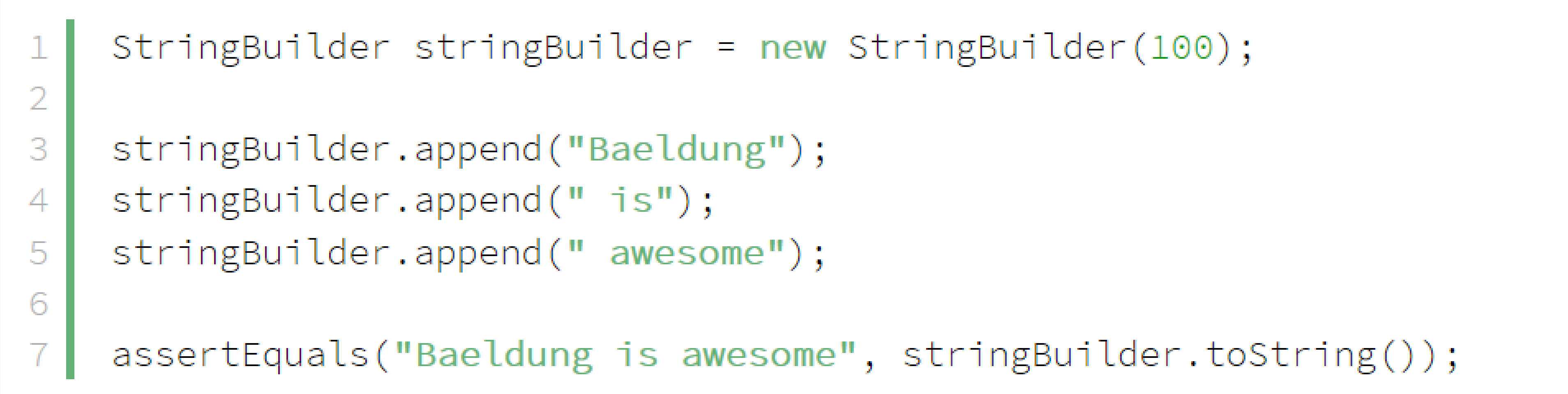
- **Комбинира стрингове от няколко реда :**

```
String quote =
"Nothing in all the world is more dangerous than " +
"sincere ignorance and conscientious stupidity.";
```

Конкатенация на стрингове

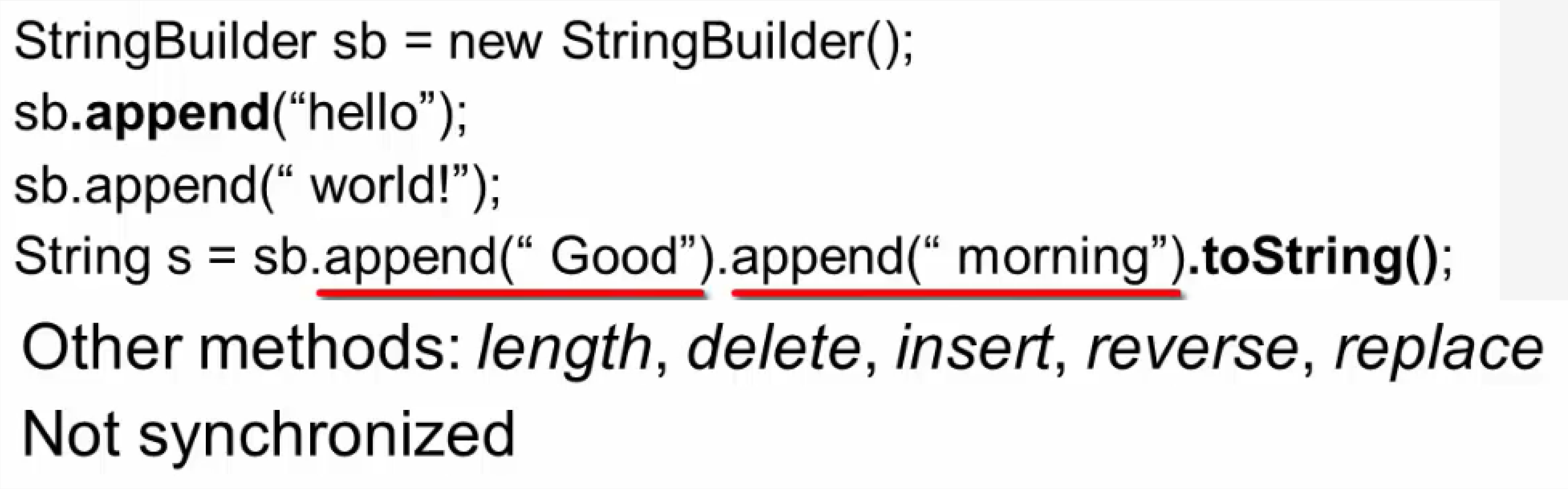
- **клас StringBuilder**
- **клас StringBuffer**

клас StringBuilder



```java
StringBuilder stringBuilder = new StringBuilder(100);

stringBuilder.append("Baeldung");
stringBuilder.append(" is");
stringBuilder.append(" awesome");

assertEquals("Baeldung is awesome", stringBuilder.toString());
```

клас StringBuilder



```
StringBuilder sb = new StringBuilder();
sb.append(“hello”);
sb.append(“ world!”);
String s = sb.append(“ Good”).append(“ morning”).toString();
```

Other methods: *length*, *delete*, *insert*, *reverse*, *replace*

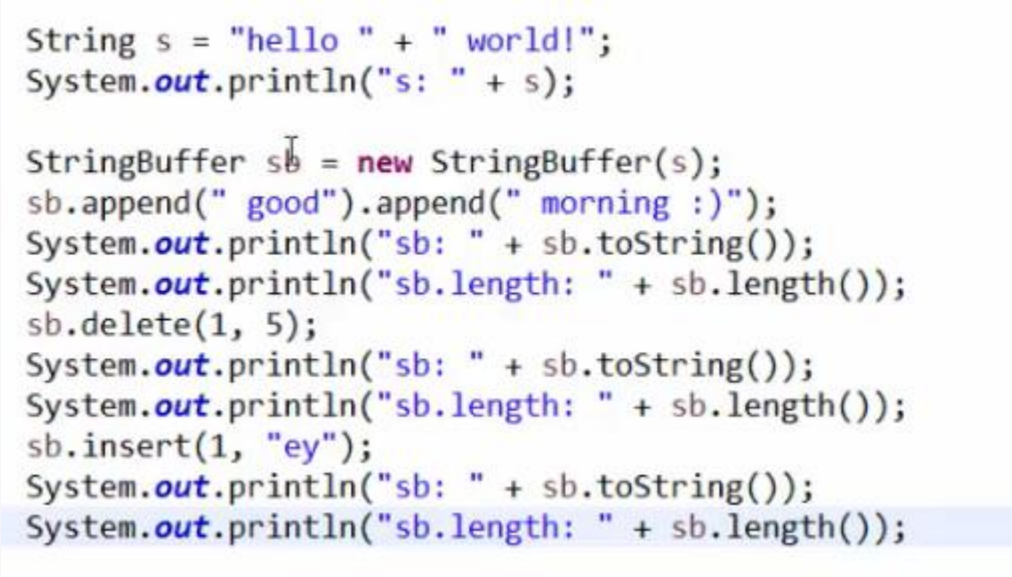
Not synchronized

клас StringBuffer

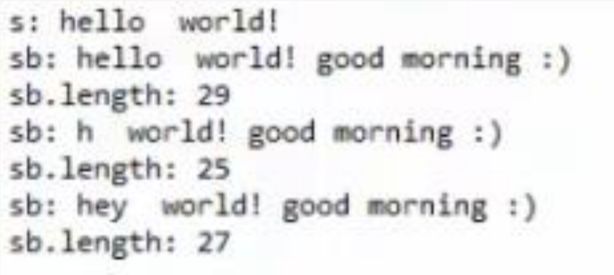
public final class **StringBuffer**
extends Object
implements Serializable, CharSequence

- StringBuffer е сингхронизиран = бавен

StringBuffer



```java
String s = "hello " + " world!";
System.out.println("s: " + s);

StringBuffer sb = new StringBuffer(s);
sb.append(" good").append(" morning :)");
System.out.println("sb: " + sb.toString());
System.out.println("sb.length: " + sb.length());
sb.delete(1, 5);
System.out.println("sb: " + sb.toString());
System.out.println("sb.length: " + sb.length());
sb.insert(1, "ey");
System.out.println("sb: " + sb.toString());
System.out.println("sb.length: " + sb.length());
```

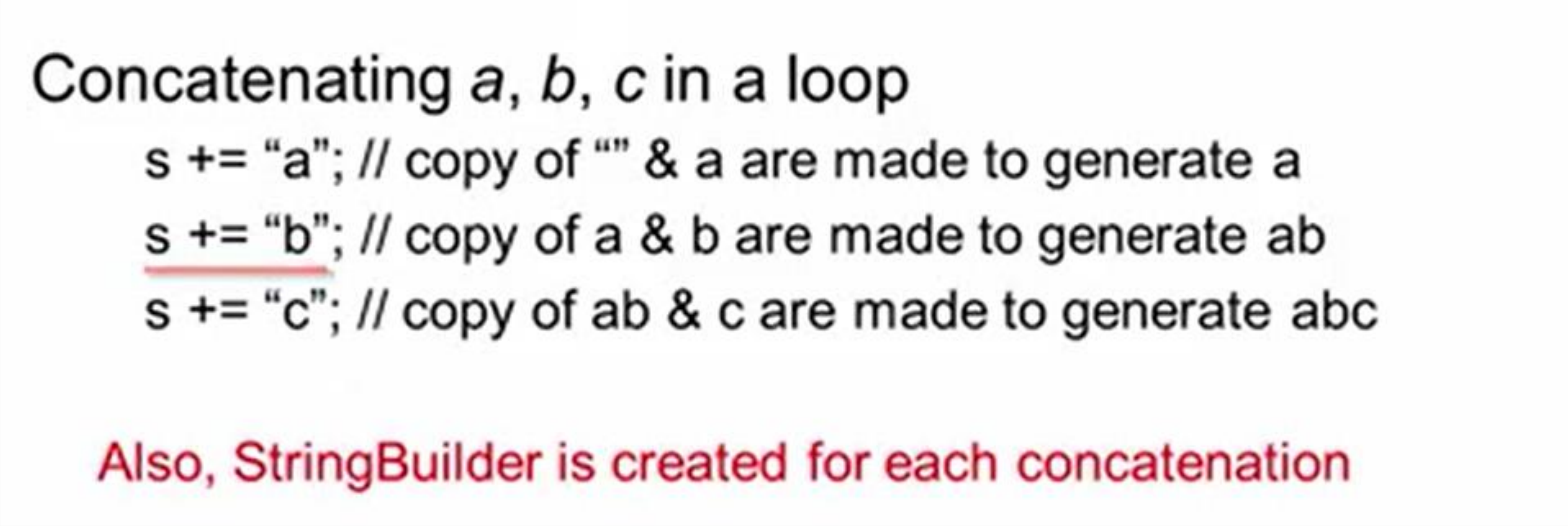


```
s: hello  world!
sb: hello  world! good morning :)
sb.length: 29
sb: h  world! good morning :)
sb.length: 25
sb: hey  world! good morning :)
sb.length: 27
```

Сравнение на всички методи за конкатенация

Оператор +

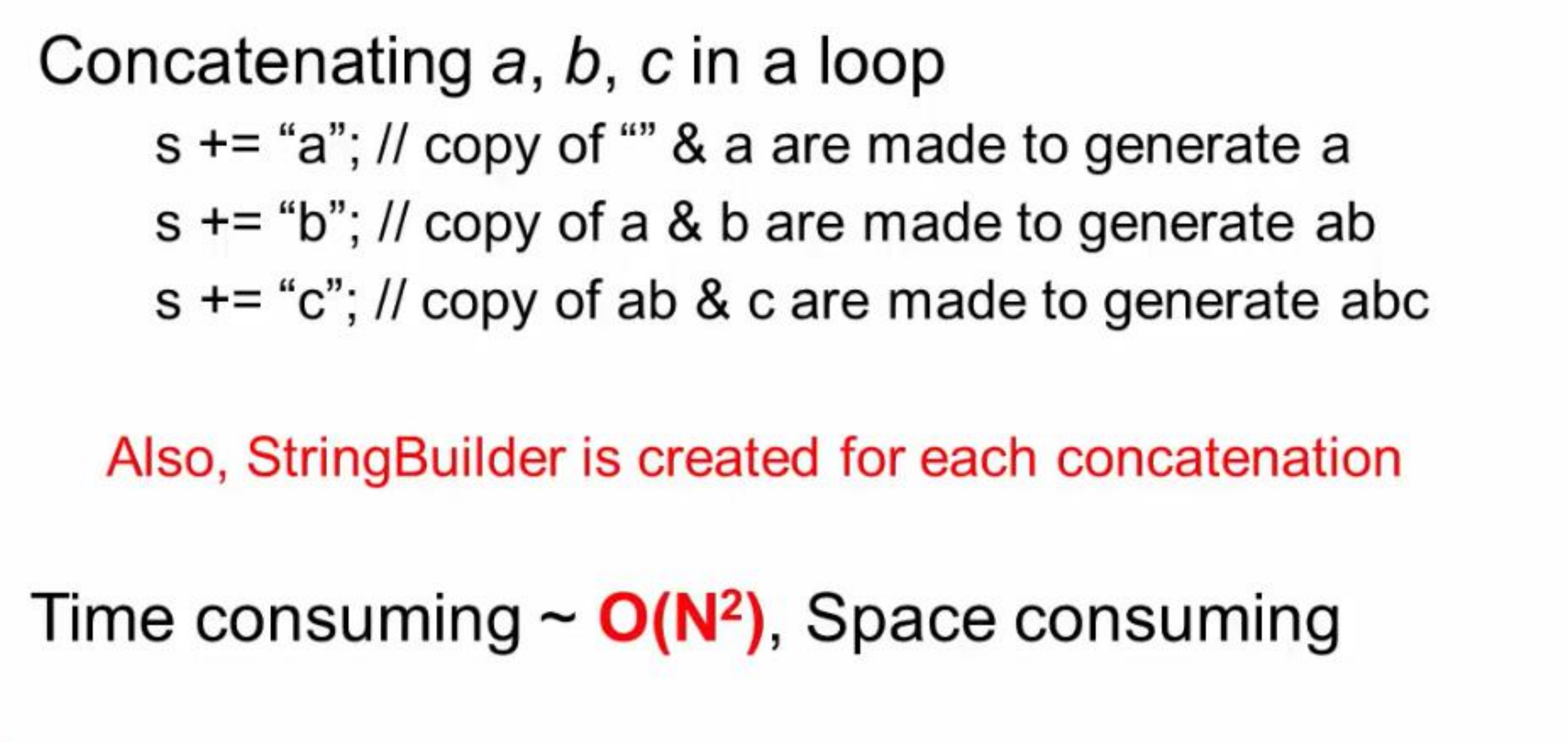
- ❑ Комбинира няколко стринга в един
- ❑ С всяка конкатенация,
  - ○ Съдържанието и на двата стринга се копират
  - ○ Нов StringBuider се създава и добавя с два стринга
  - ○ Връща стринг чрез toString()



Concatenating *a*, *b*, *c* in a loop

```
s += “a”; // copy of “” & a are made to generate a
s += “b”; // copy of a & b are made to generate ab
s += “c”; // copy of ab & c are made to generate abc
```

Also, StringBuilder is created for each concatenation



Concatenating *a*, *b*, *c* in a loop

s += “a”; // copy of “” & a are made to generate a

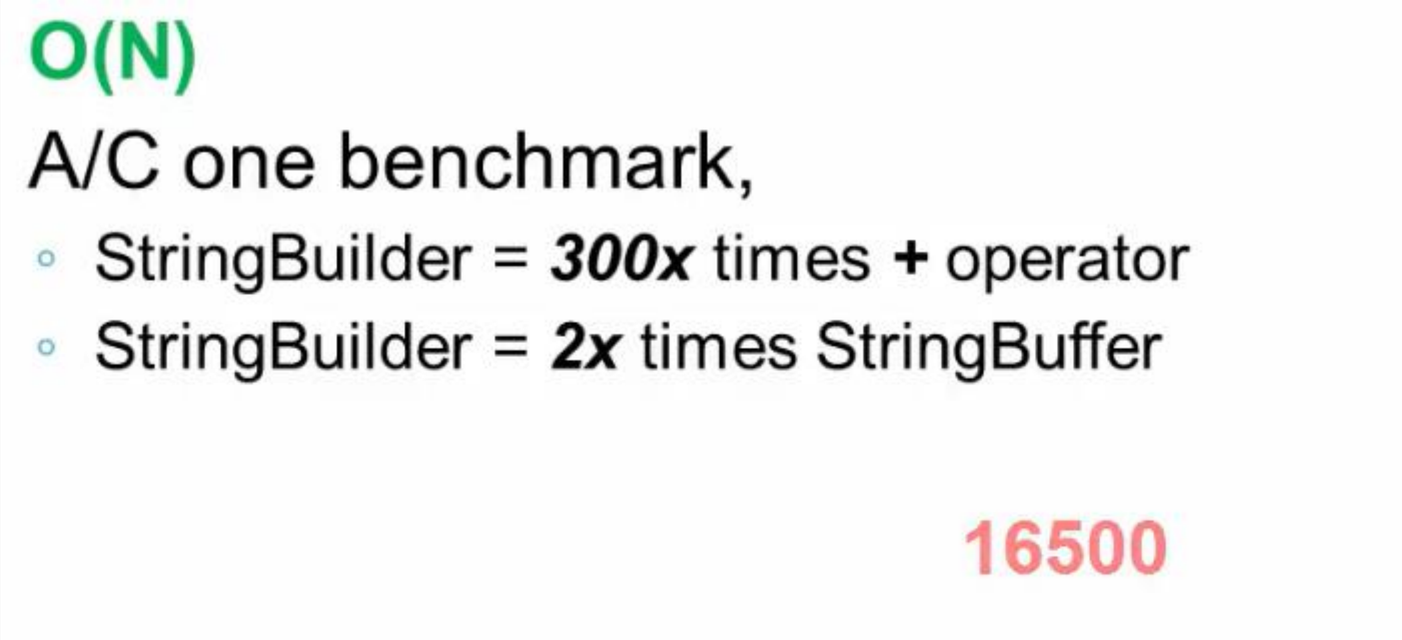
s += “b”; // copy of a & b are made to generate ab

s += “c”; // copy of ab & c are made to generate abc

Also, StringBuilder is created for each concatenation

Time consuming ~ **$O(N^2)$**, Space consuming

Use StringBuilder



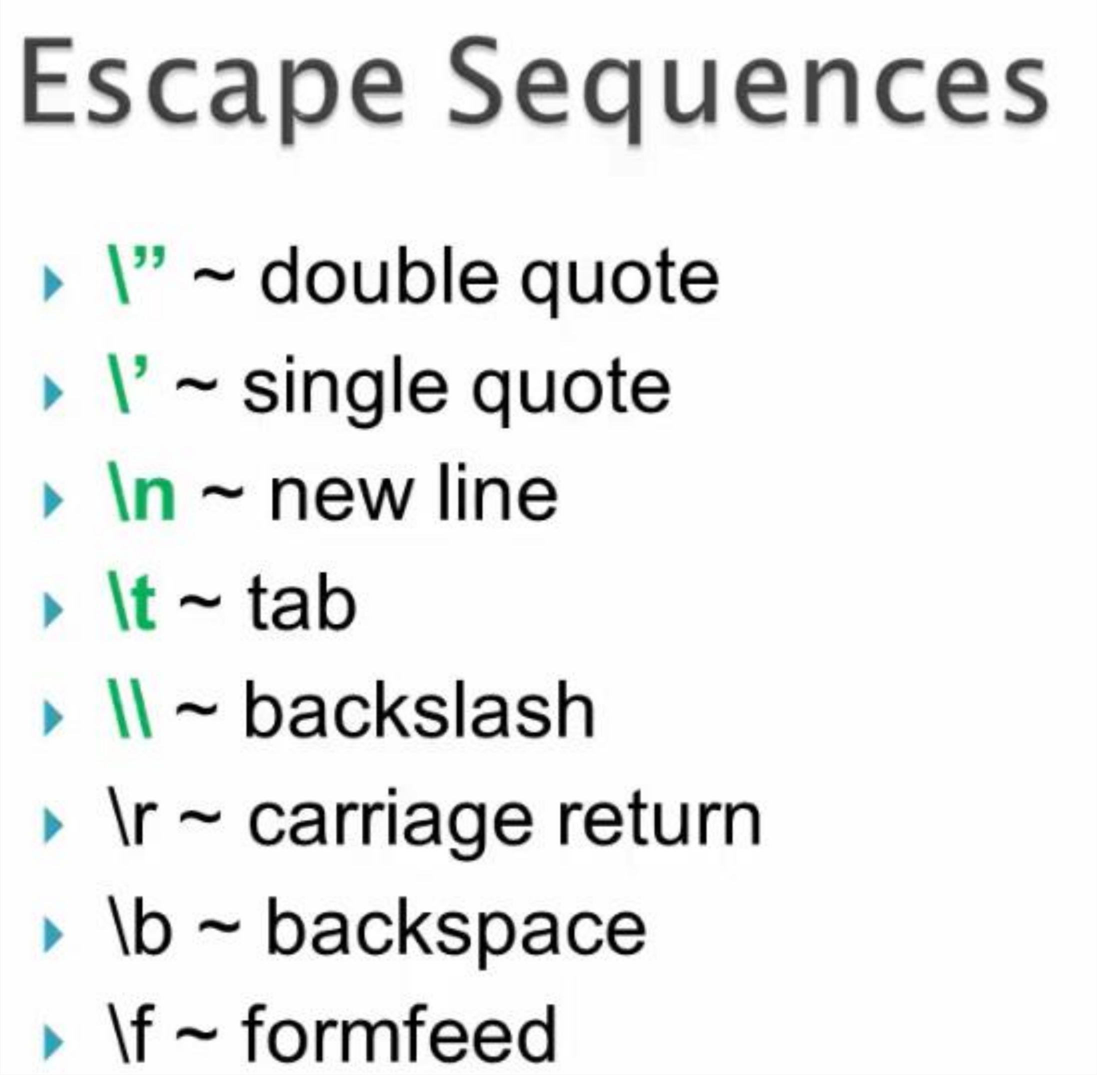
O(N)

A/C one benchmark,

- StringBuilder = ***300x*** times **+** operator
- StringBuilder = ***2x*** times StringBuffer

16500

https://www.ntu.edu.sg/home/ehchua/programming/java/J3d_String.html#zz-3.3

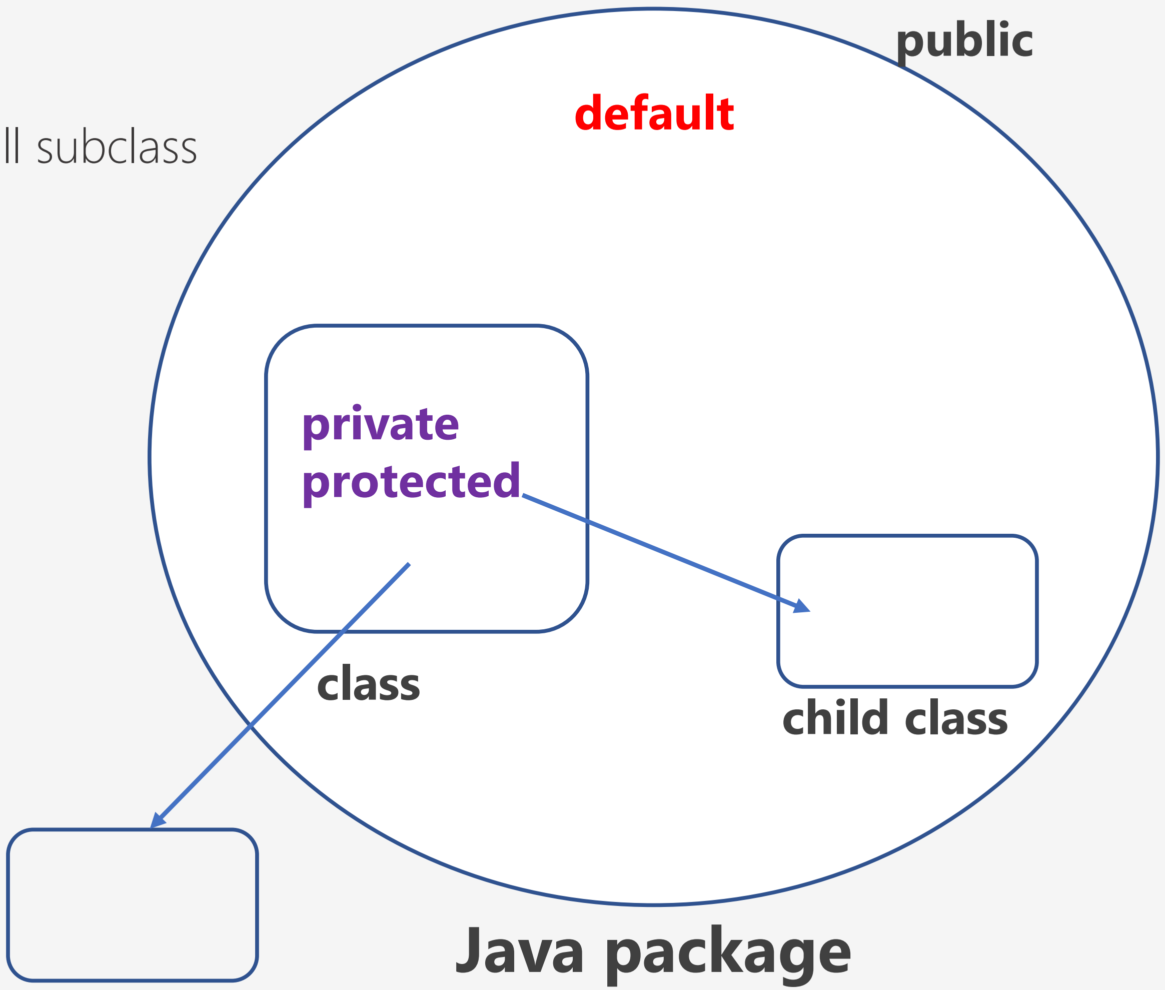


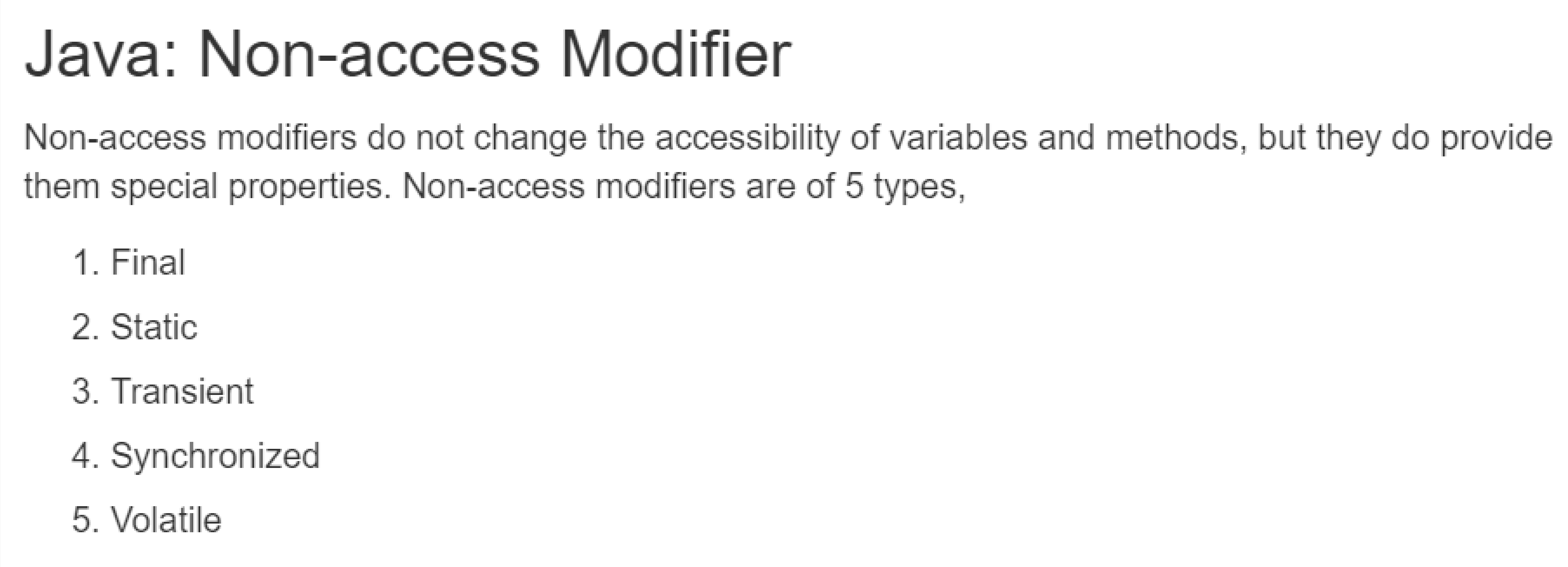
Escape Sequences

- **\”** ~ double quote
- **\’** ~ single quote
- **\n** ~ new line
- **\t** ~ tab
- \ ~ backslash
- \r ~ carriage return
- \b ~ backspace
- \f ~ formfeed

Access Control Modifiers

Default: Default has scope only inside the same package
Public: Public has scope that is visible everywhere
Protected: Protected has scope within the package and all subclass
Private: Private has scope only within the classes



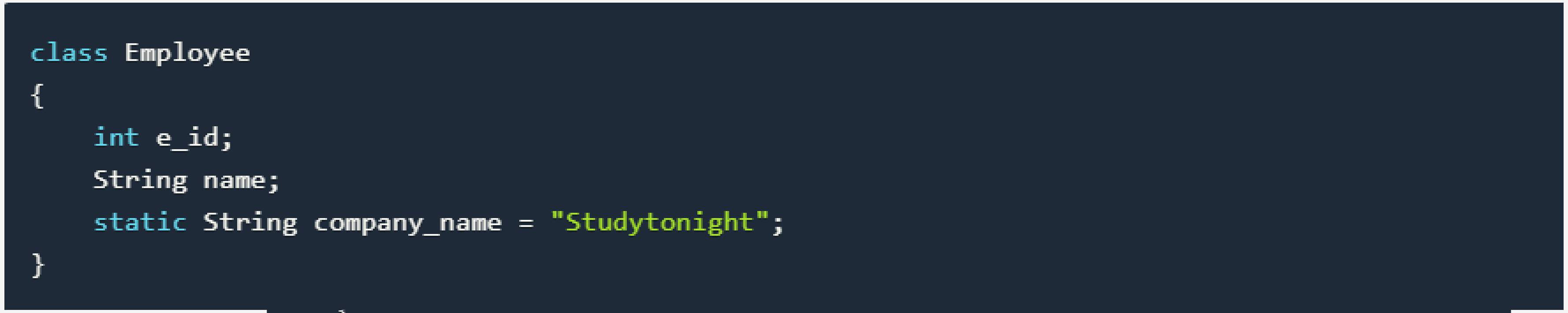


Java: Non-access Modifier

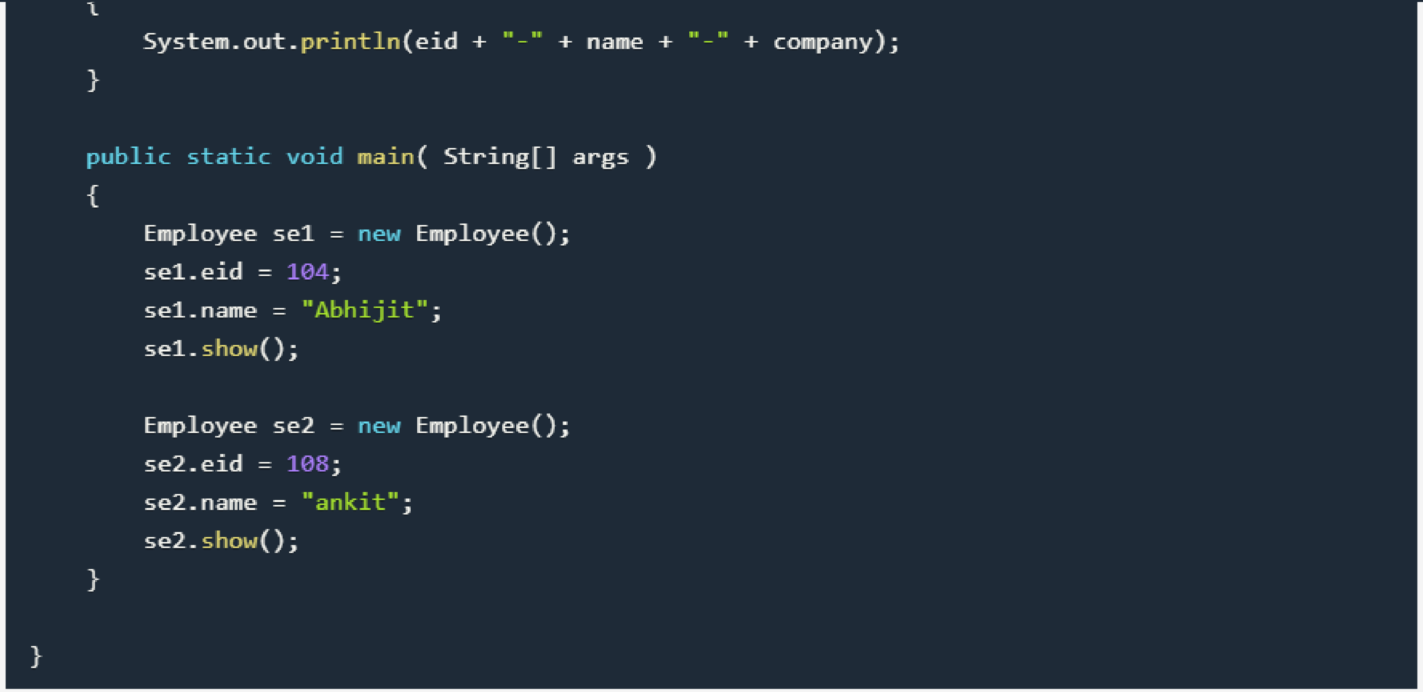
Non-access modifiers do not change the accessibility of variables and methods, but they do provide them special properties. Non-access modifiers are of 5 types,

1. Final
2. Static
3. Transient
4. Synchronized
5. Volatile

Static Modifier

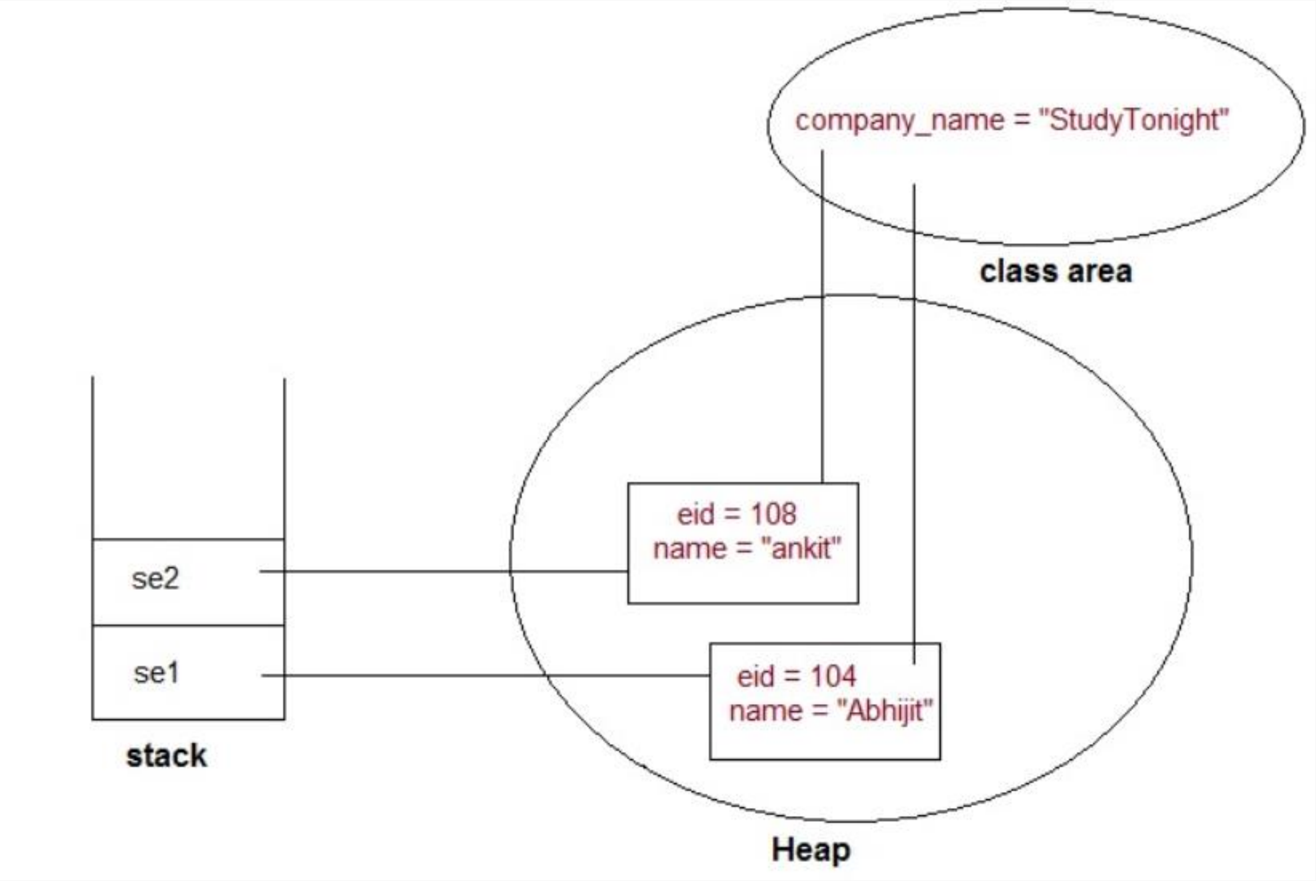


```java
class Employee
{
    int e_id;
    String name;
    static String company_name = "Studytonight";
}
```

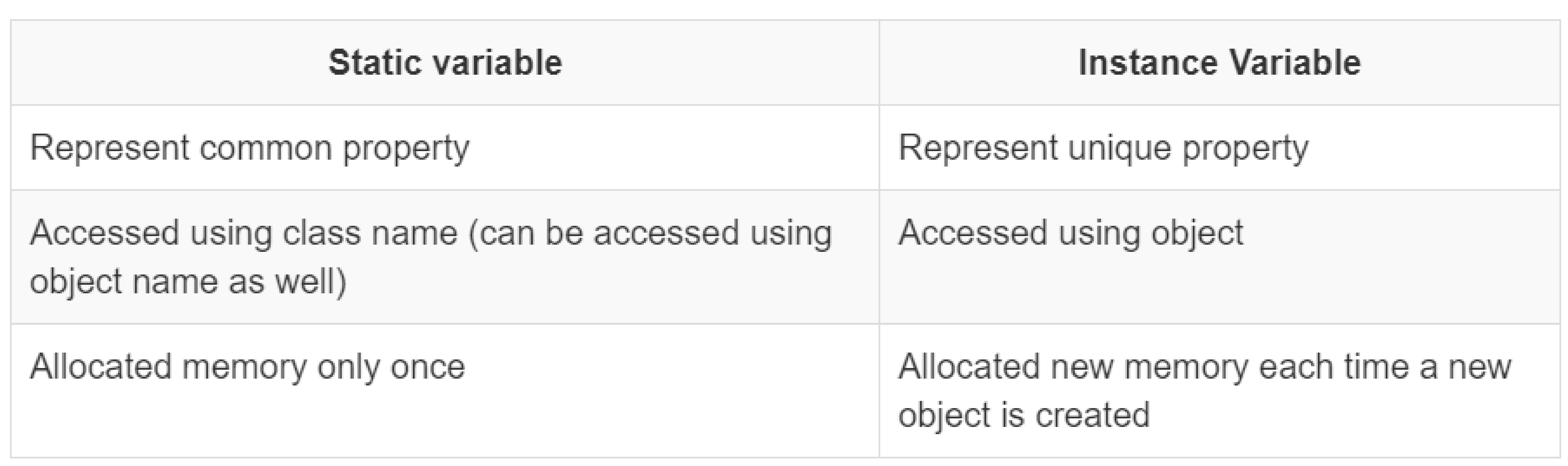


```java
    {
        System.out.println(eid + "-" + name + "-" + company);
    }

    public static void main( String[] args )
    {
        Employee se1 = new Employee();
        se1.eid = 104;
        se1.name = "Abhijit";
        se1.show();

        Employee se2 = new Employee();
        se2.eid = 108;
        se2.name = "ankit";
        se2.show();
    }

}
```

Static Modifier

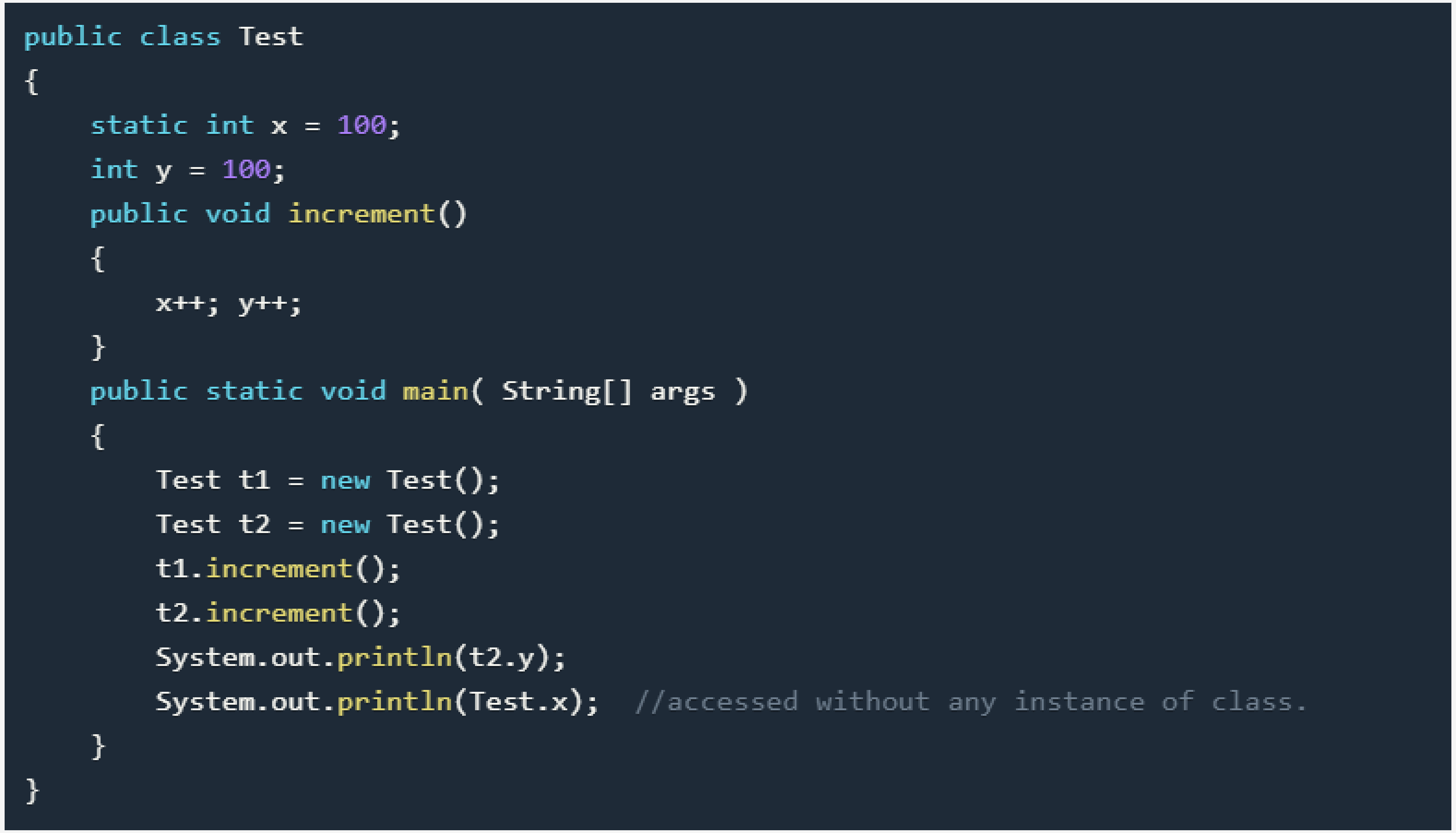


Static variable vs Instance variable



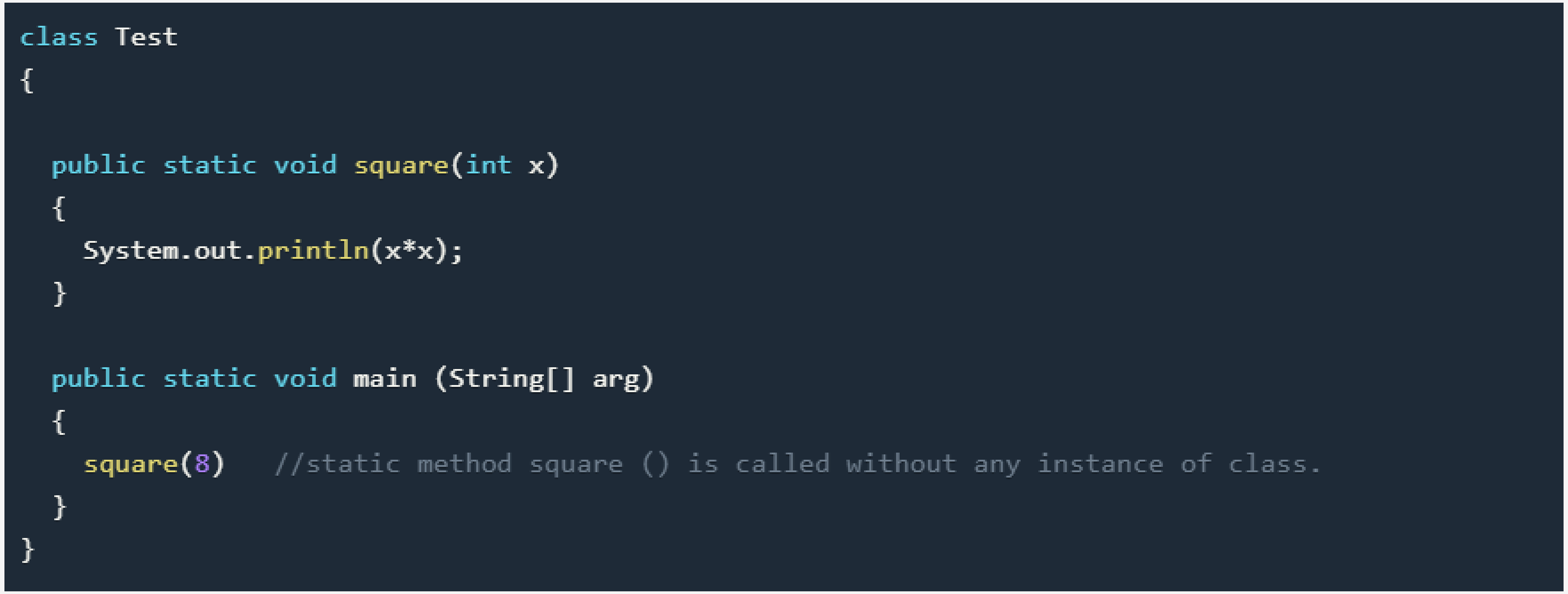
Static variable	Instance Variable
Represent common property	Represent unique property
Accessed using class name (can be accessed using object name as well)	Accessed using object
Allocated memory only once	Allocated new memory each time a new object is created

Static variable vs Instance variable

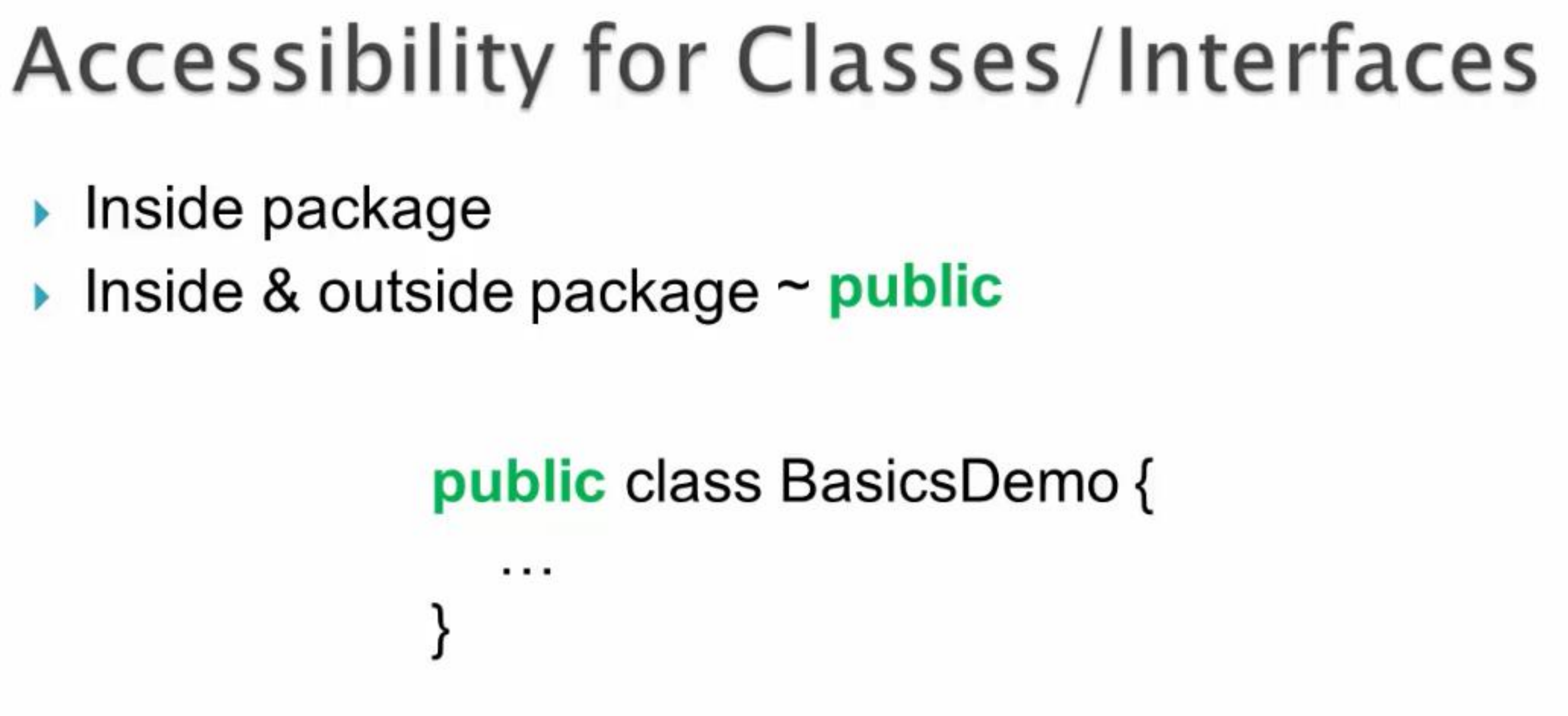


```
public class Test
{
    static int x = 100;
    int y = 100;
    public void increment()
    {
        x++; y++;
    }
    public static void main( String[] args )
    {
        Test t1 = new Test();
        Test t2 = new Test();
        t1.increment();
        t2.increment();
        System.out.println(t2.y);
        System.out.println(Test.x);  //accessed without any instance of class.
    }
}
```

Static Method in Java



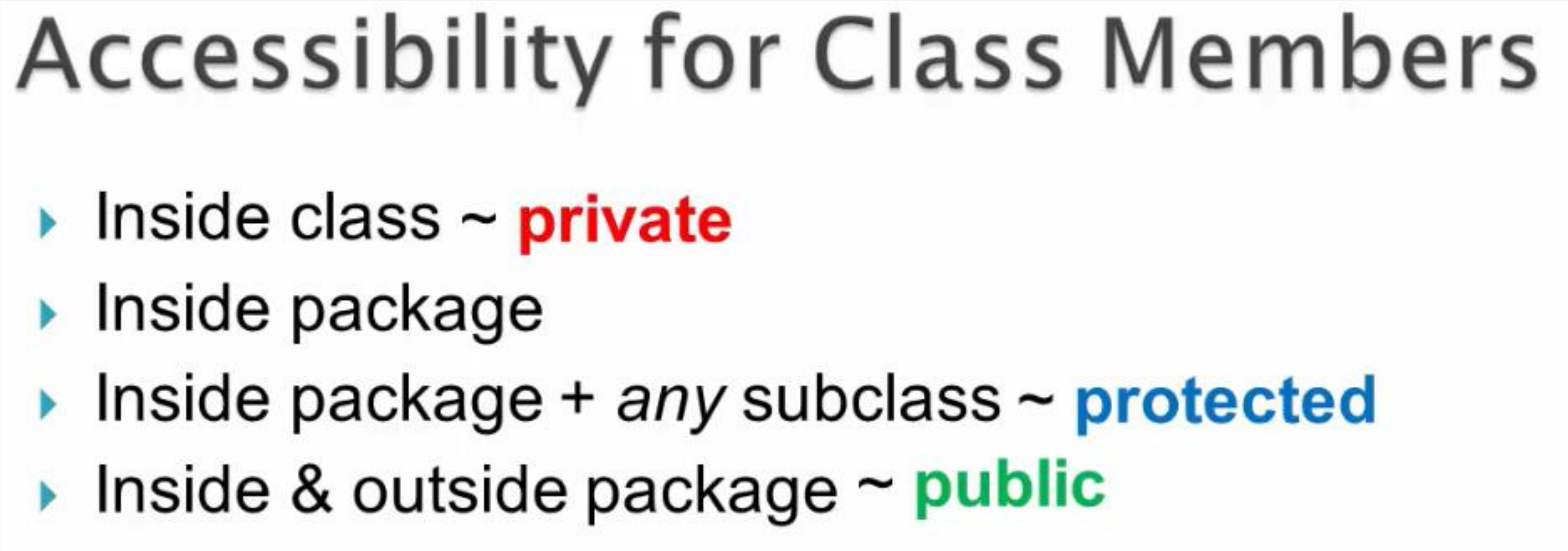
```
class Test
{

  public static void square(int x)
  {
    System.out.println(x*x);
  }

  public static void main (String[] arg)
  {
    square(8)   //static method square () is called without any instance of class.
  }
}
```



Accessibility for Classes/Interfaces

- Inside package
- Inside & outside package ~ **public**

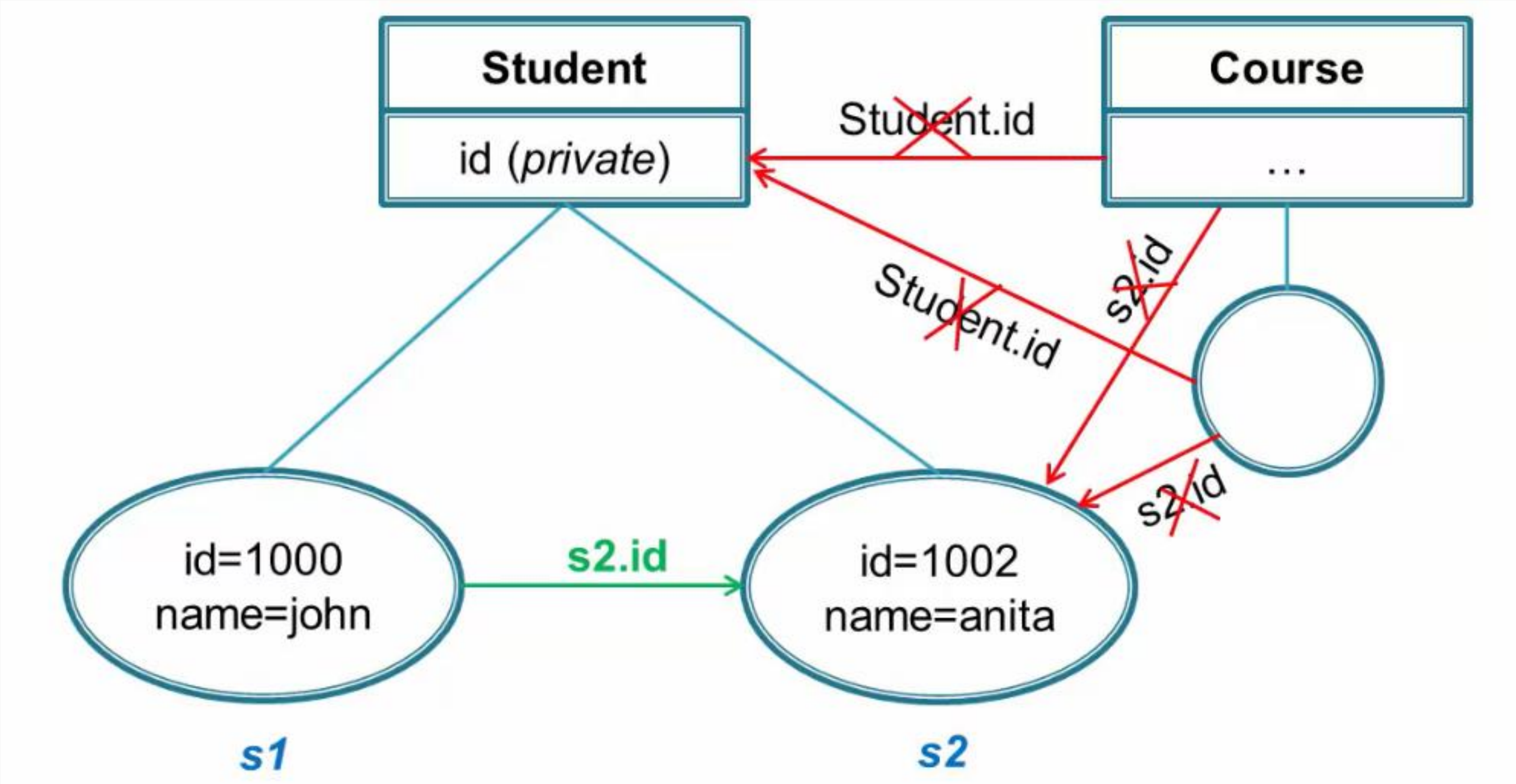
```
public class BasicsDemo {
    ...
}
```



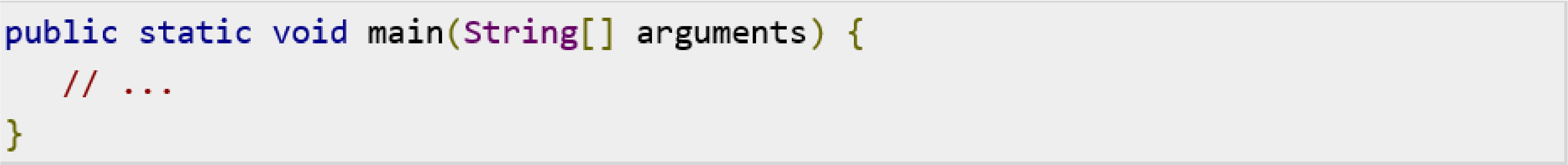
Accessibility for Class Members

- Inside class ~ **private**
- Inside package
- Inside package + *any* subclass ~ **protected**
- Inside & outside package ~ **public**

Private Access Modifier

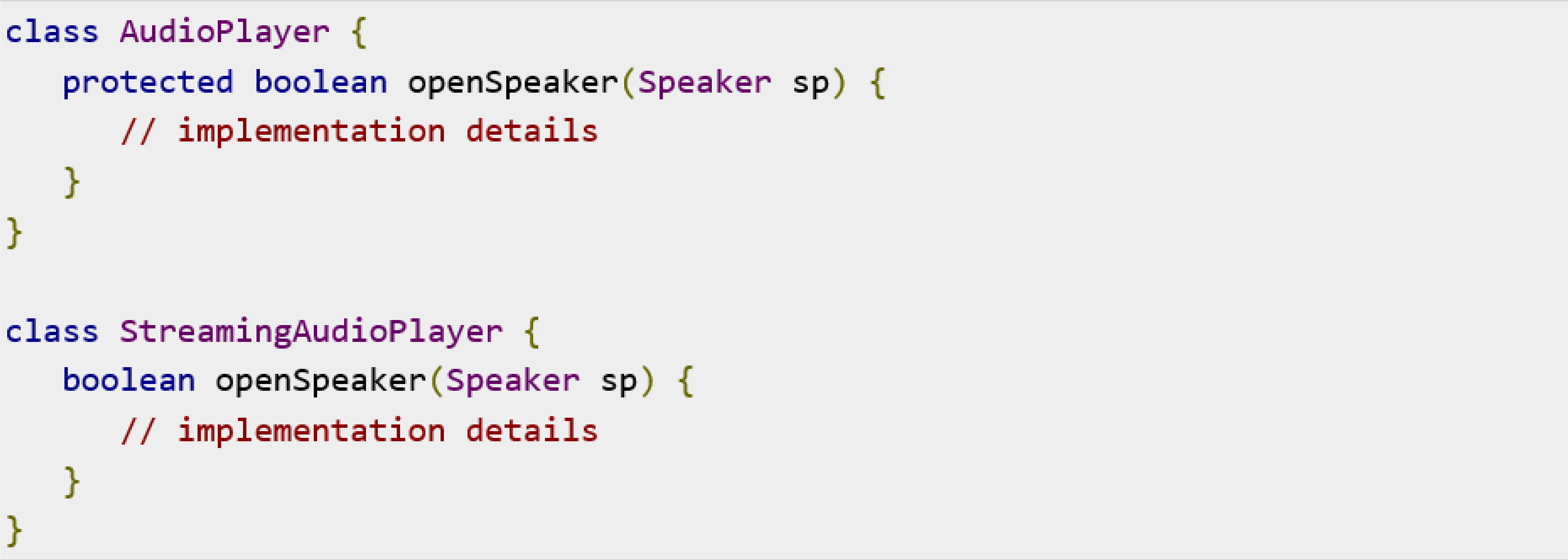


Public Access Modifier



```
public static void main(String[] arguments) {
   // ...
}
```

Protected Access Modifier



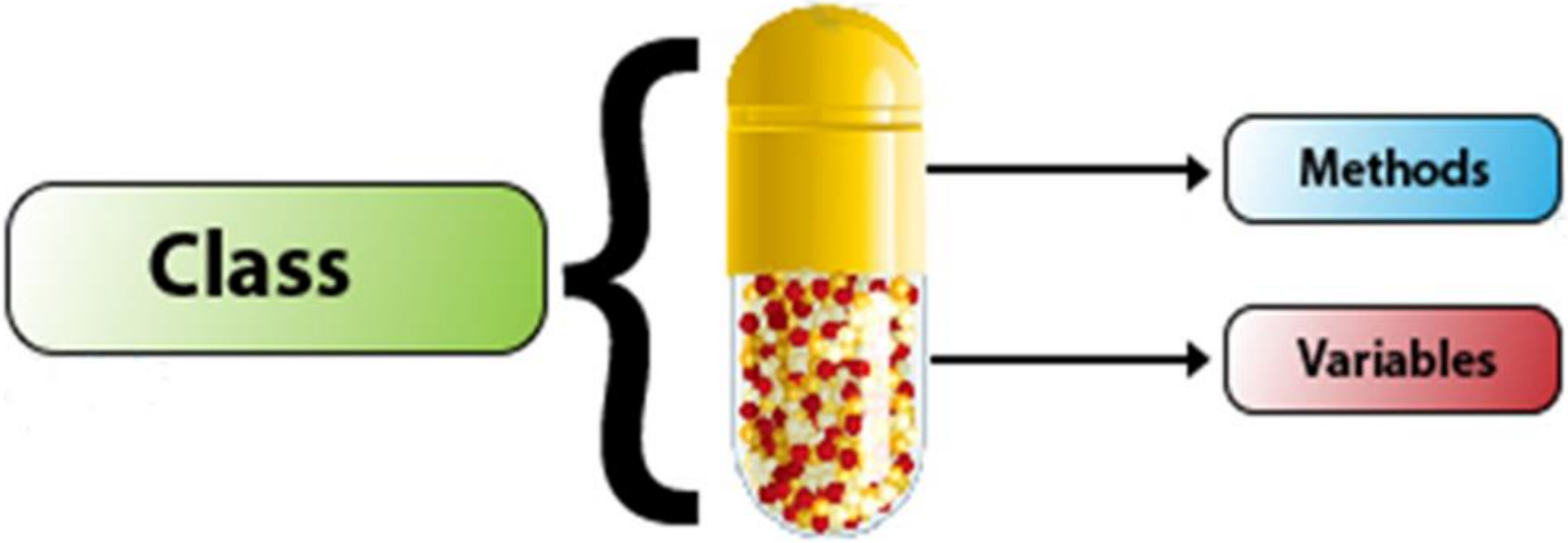
```
class AudioPlayer {
   protected boolean openSpeaker(Speaker sp) {
      // implementation details
   }
}

class StreamingAudioPlayer {
   boolean openSpeaker(Speaker sp) {
      // implementation details
   }
}
```

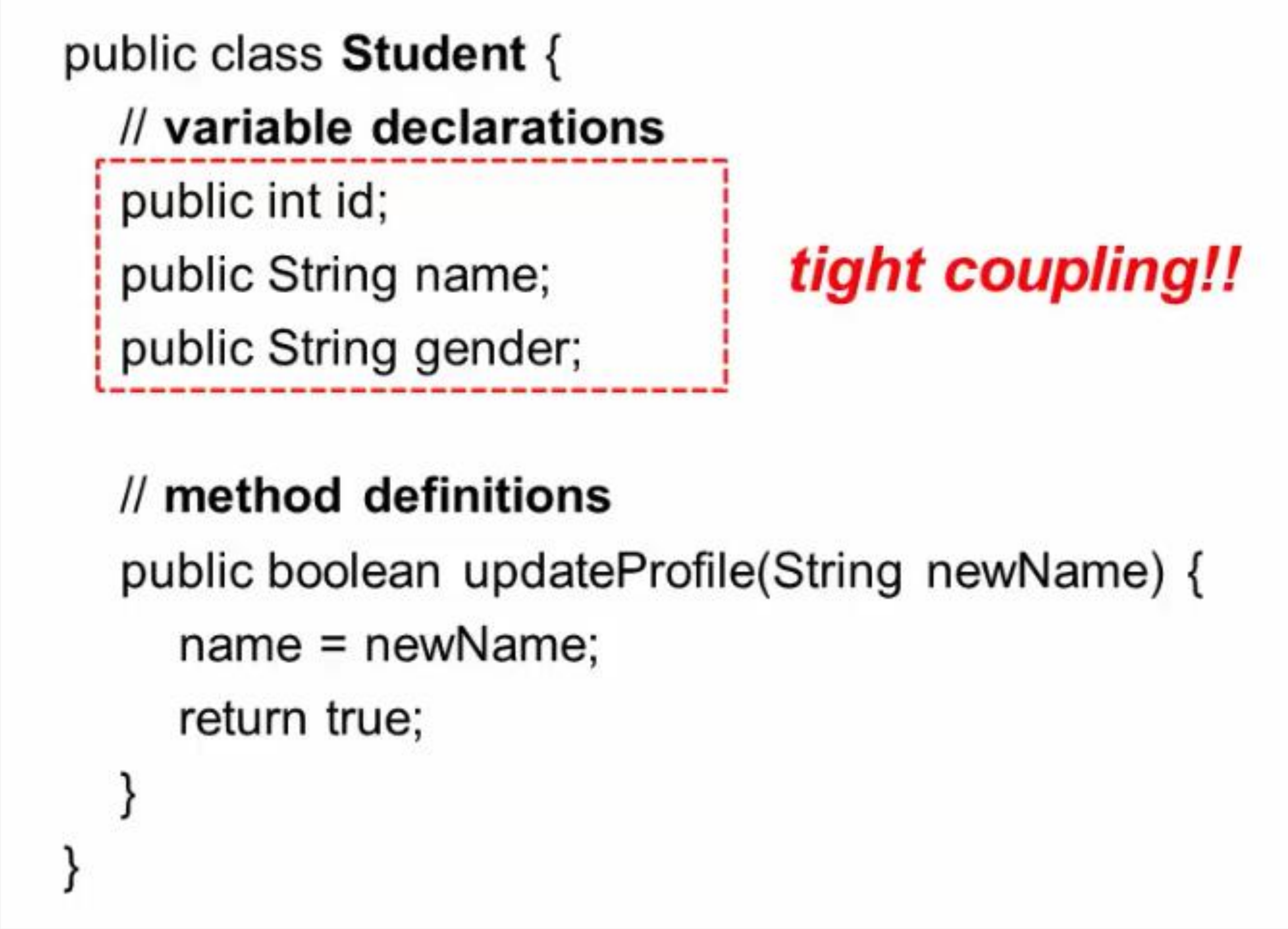
Достъп и наследяване

- методи, които са public в суперкласа също трябва да бъдат декларирани public в подкласовете;
- Методи декларирани protected в суперкласа, трябва да бъдат или protected или public в подкласовете; не мога да бъдат private;
- Методи декларирани private не се наследяват въобще

Скриване на информация

Encapsulation in Java





```
public class Student {
   // variable declarations
   public int id;
   public String name;
   public String gender;

   // method definitions
   public boolean updateProfile(String newName) {
      name = newName;
      return true;
   }
}
```

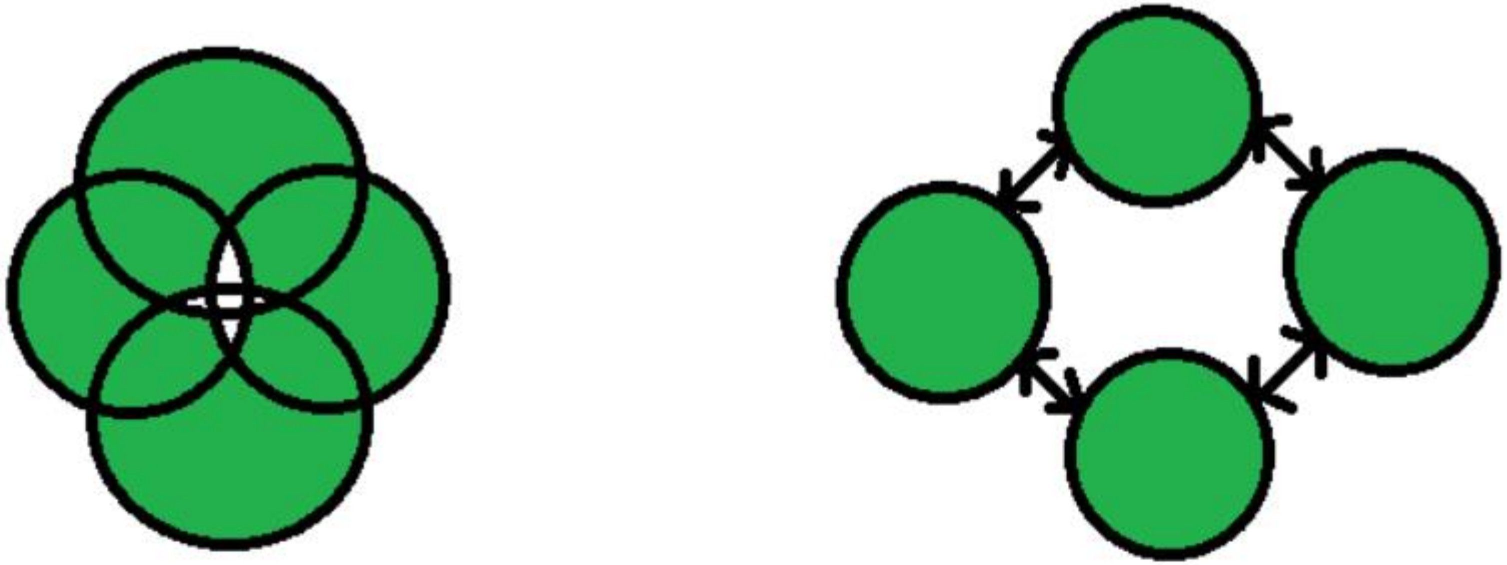
tight coupling!!

Tightly coupling

```
// Java program to illustrate
// tight coupling concept
class Subject {
        Topic t = new Topic();
        public void startReading()
        {
                t.understand();
        }
}
class Topic {
        public void understand()
        {
                System.out.println("Tight coupling
concept");
        }
}
```

Loose coupling

```
public interface Topic
{
        void understand();
}
class Topic1 implements Topic {
public void understand()
        {
                System.out.println("Got it");
        }
} class Topic2 implements Topic {
public void unserstand()
        {
                System.out.println("understand");
        }
} public class Subject {
public static void main(String[] args)
        {
                Topic t = new Topic1();
                t.understand();
        }
}
```



Tight coupling:

1. More Interdependency
2. More coordination
3. More information flow

Loose coupling:

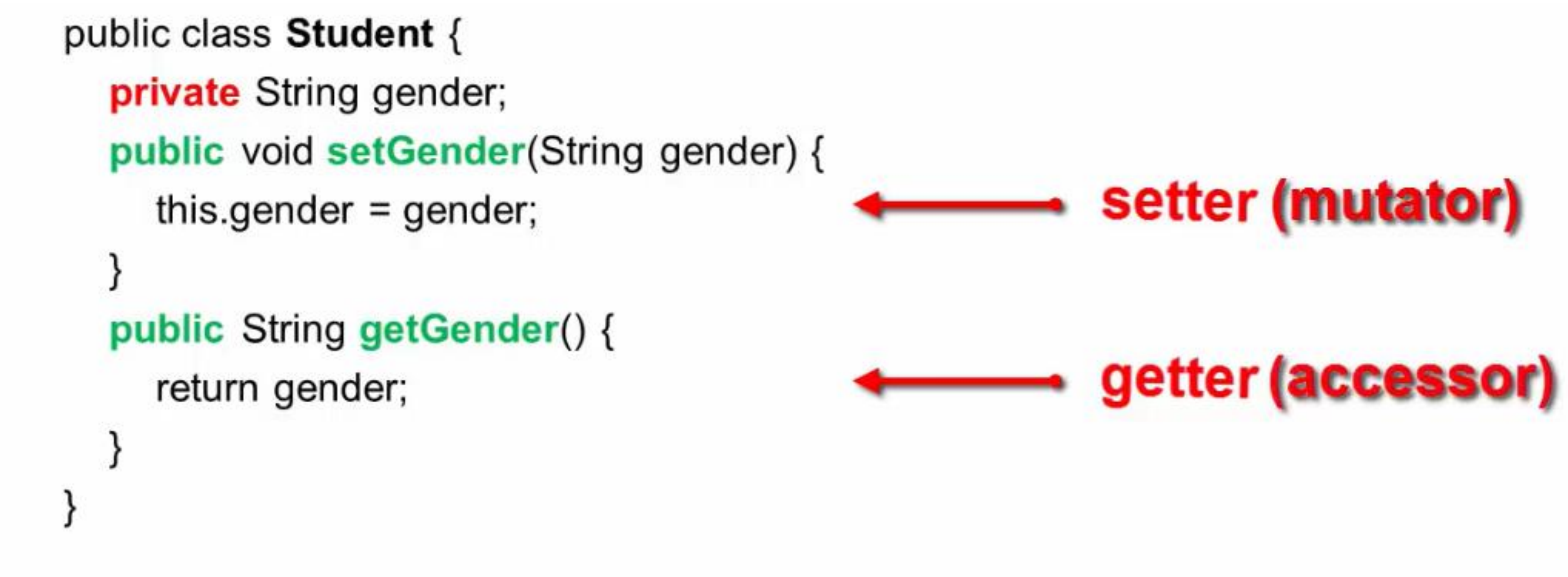
1. Less Interdependency
2. Less coordination
3. Less information flow

```
public class Student {
   // variable declarations
   public int id;
   public String name;
   public String gender;

   // method definitions
   public boolean updateProfile(String newName) {
      name = newName;
      return true;
   }
}
```

tight coupling!!

Information Hiding

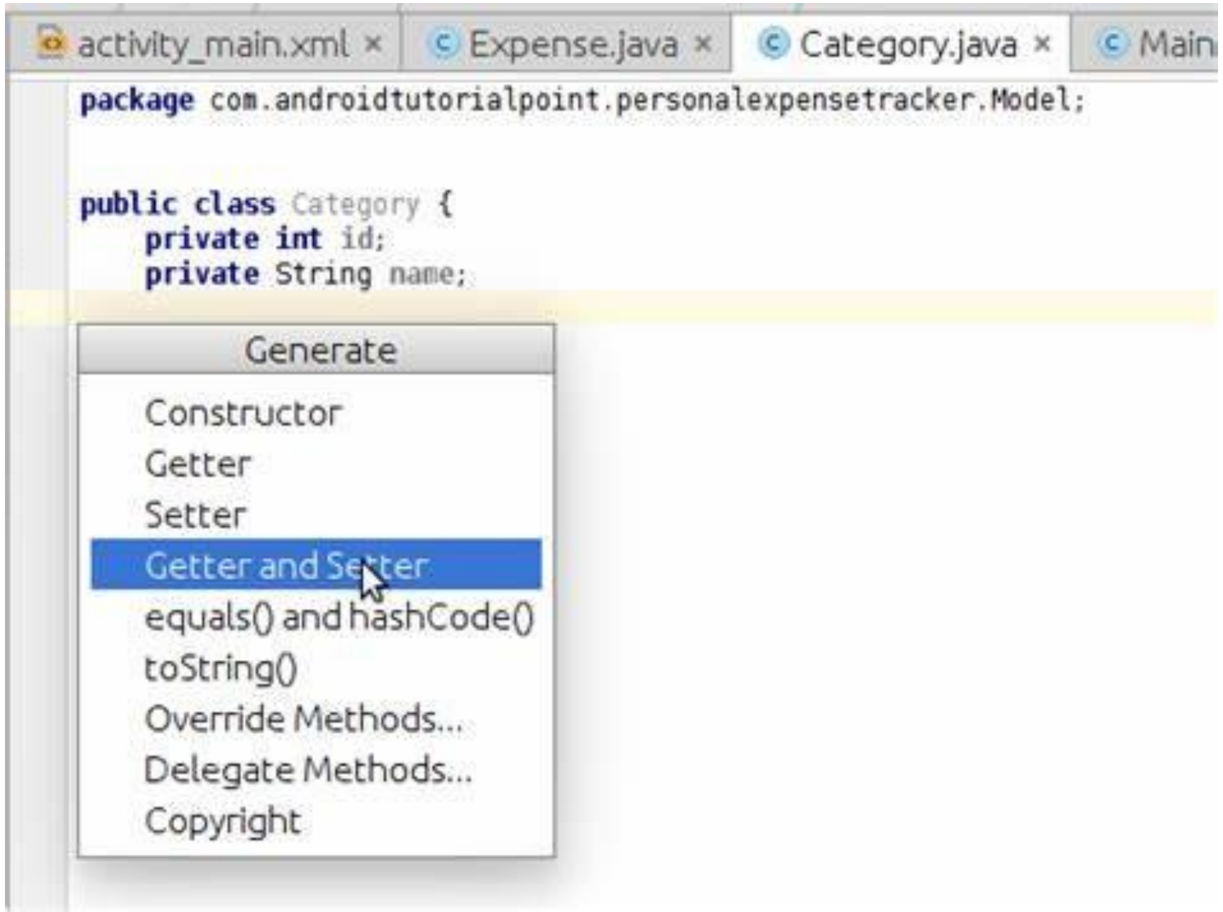
Item 14: In public classes, use accessor methods, not public fields



```java
public class Student {
    private String gender;
    public void setGender(String gender) {
        this.gender = gender;
    }
    public String getGender() {
        return gender;
    }
}
```

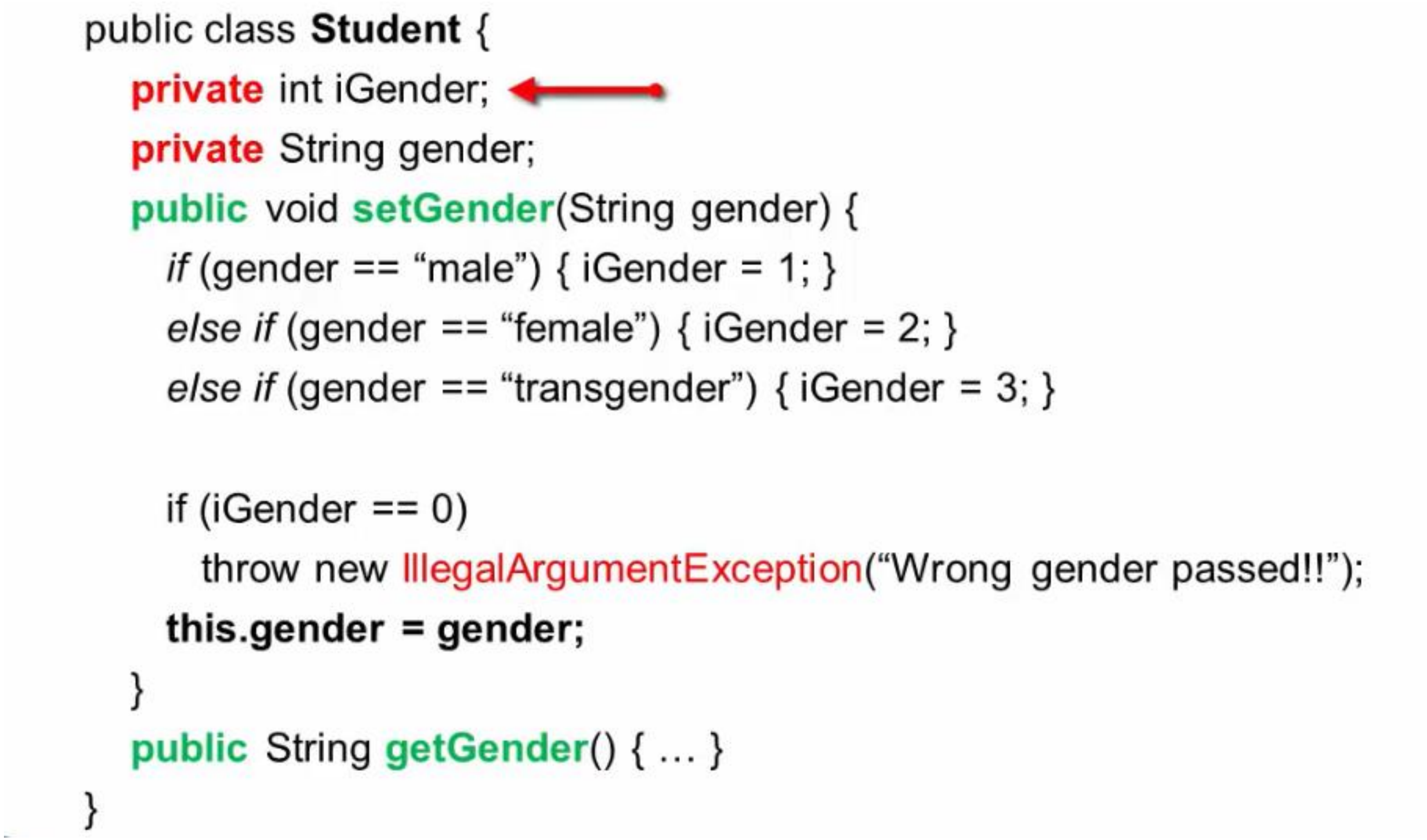
setter (mutator)

getter (accessor)



```
package com.androidtutorialpoint.personalexpensetracker.Model;


public class Category {
    private int id;
    private String name;
```

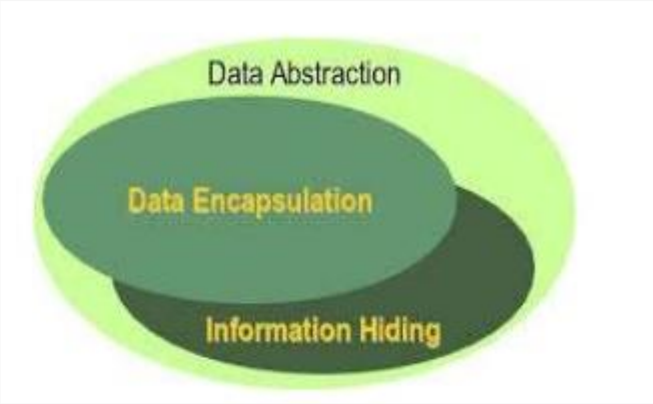
```
public class Student {
  private String gender;
  public void setGender(String gender) {
    if (gender.equals(“male”) || gender.equals(“female”)
                              || gender.equals(“transgender”)) {
        this.gender = gender;
    } else {
→     throw new IllegalArgumentException(“Wrong gender passed!!”);
    }
  }
  public String getGender() { … }
}
```



```
public class Student {
   private int iGender;
   private String gender;
   public void setGender(String gender) {
     if (gender == “male”) { iGender = 1; }
     else if (gender == “female”) { iGender = 2; }
     else if (gender == “transgender”) { iGender = 3; }

     if (iGender == 0)
       throw new IllegalArgumentException(“Wrong gender passed!!”);
     this.gender = gender;
   }
   public String getGender() { … }
}
```

Скриване на информацията

Абстракция на данните (скриване на информацията) представлява процеса, който осигурява само необходимата информация извън класа/интерфейса и скрива другите детайли.



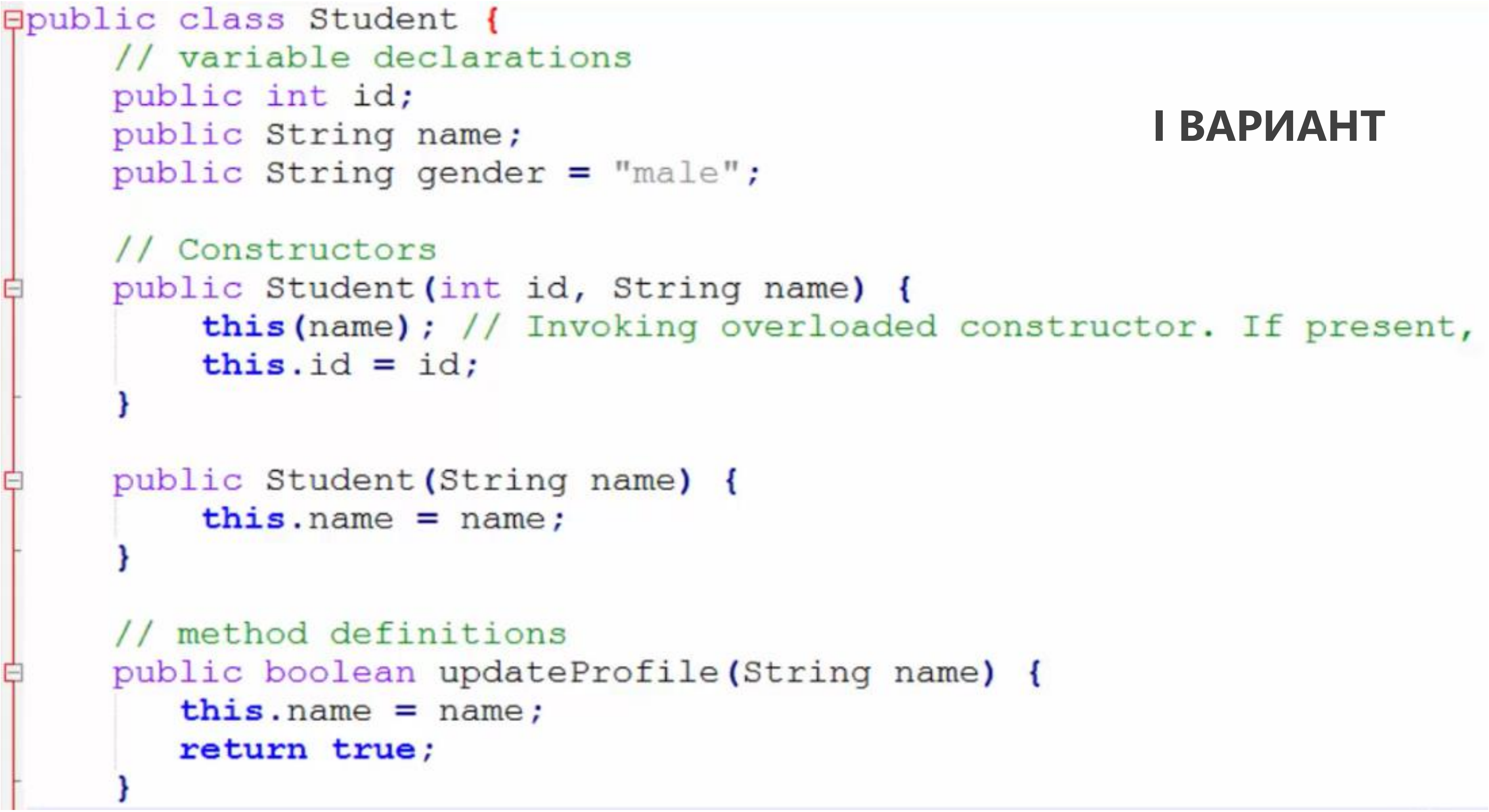
Item 13: Accessibility for Class Members

- Design ***minimal public API*** of your class
- Make all other members ***private***
- Make a member **default**, only if ***really*** needed
  - Frequent changes implies *reexamine your design!!*

Достъпността на класовете/интерфейсите

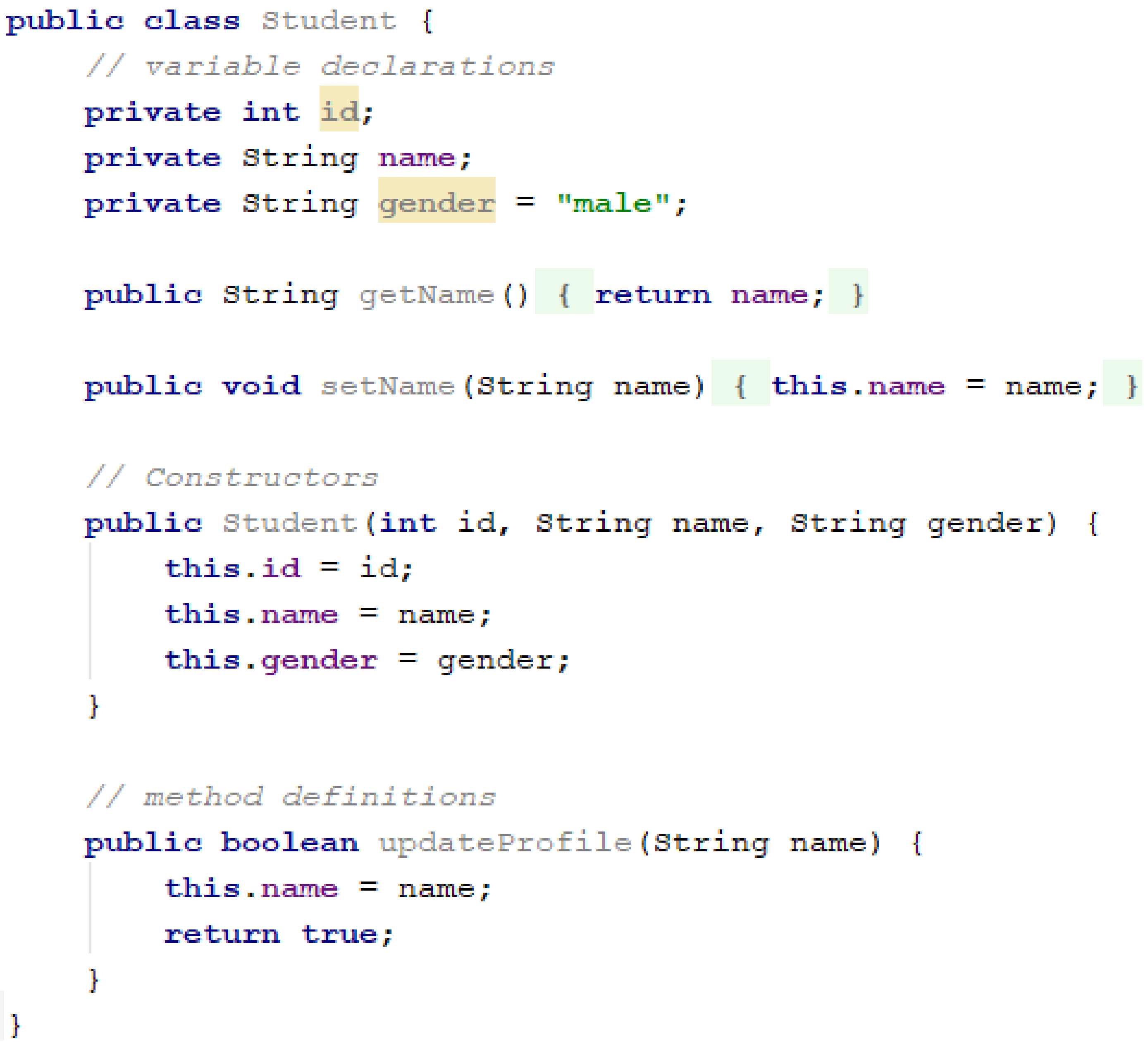
- Когато е възможно полетата и моетодите да бъдат **default**
- Когато е възможно само един клас да се наследява, използвайте private nested.

I ВАРИАНТ

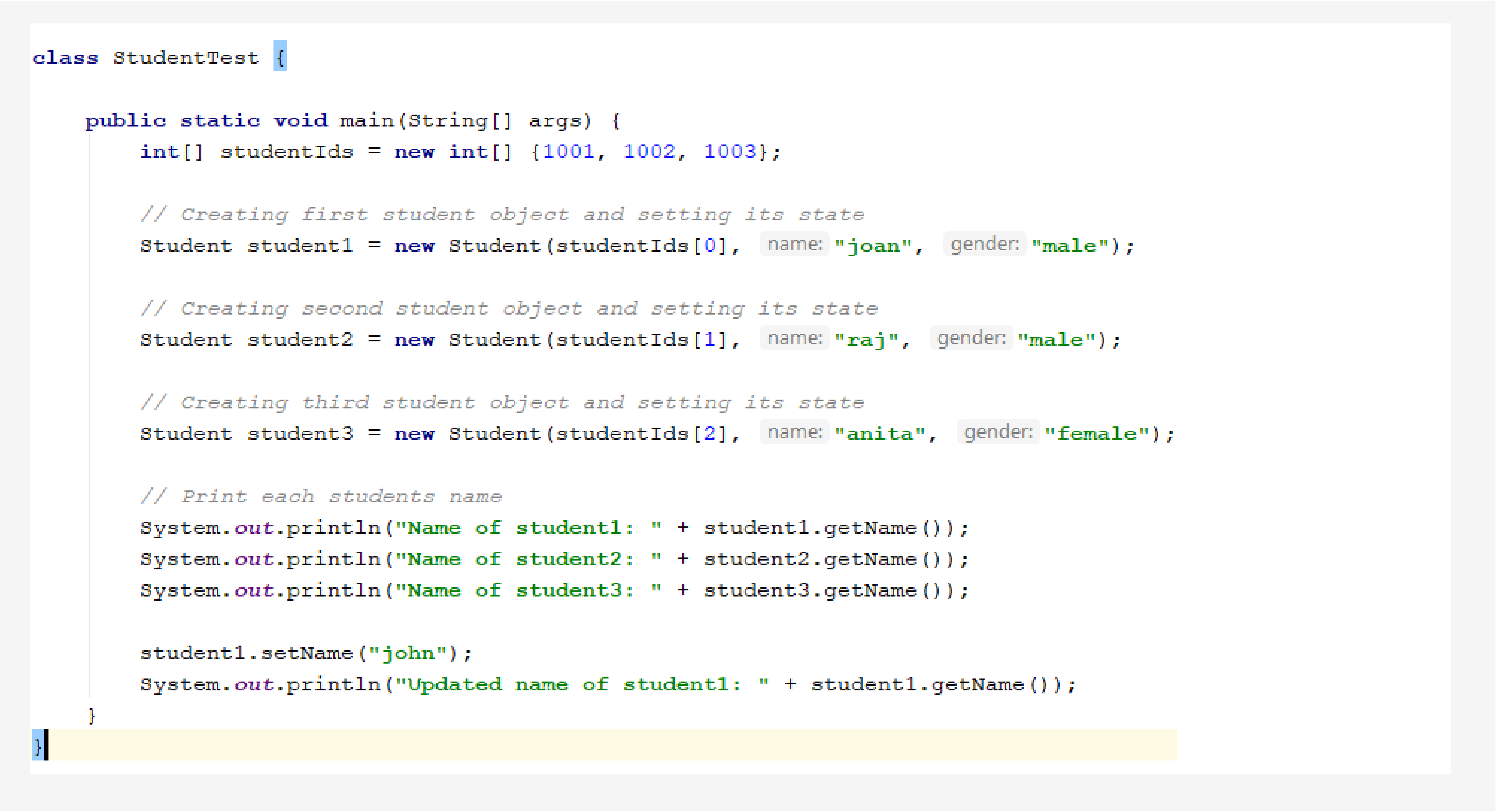


```
public class Student {
    // variable declarations
    public int id;
    public String name;
    public String gender = "male";

    // Constructors
    public Student(int id, String name) {
        this(name); // Invoking overloaded constructor. If present,
        this.id = id;
    }

    public Student(String name) {
        this.name = name;
    }

    // method definitions
    public boolean updateProfile(String name) {
       this.name = name;
       return true;
    }
```

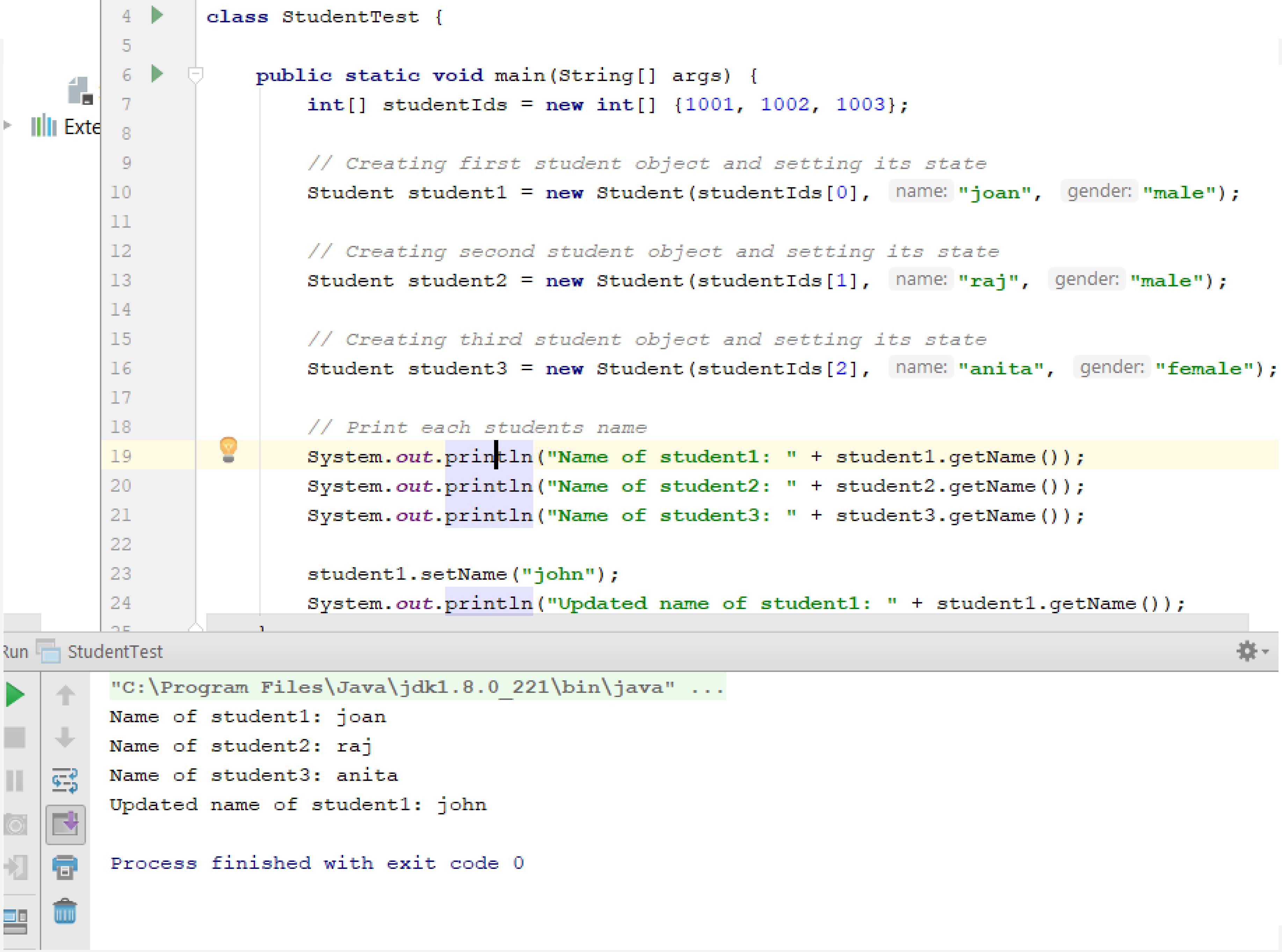
II ВАРИАНТ



```java
public class Student {
    // variable declarations
    private int id;
    private String name;
    private String gender = "male";

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    // Constructors
    public Student(int id, String name, String gender) {
        this.id = id;
        this.name = name;
        this.gender = gender;
    }

    // method definitions
    public boolean updateProfile(String name) {
        this.name = name;
        return true;
    }
}
```



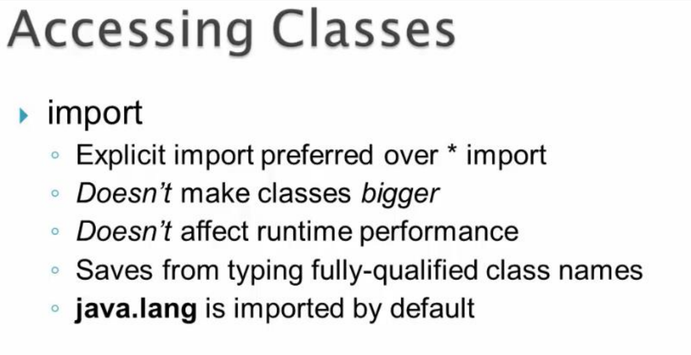
```
class StudentTest {

    public static void main(String[] args) {
        int[] studentIds = new int[] {1001, 1002, 1003};

        // Creating first student object and setting its state
        Student student1 = new Student(studentIds[0], name: "joan", gender: "male");

        // Creating second student object and setting its state
        Student student2 = new Student(studentIds[1], name: "raj", gender: "male");

        // Creating third student object and setting its state
        Student student3 = new Student(studentIds[2], name: "anita", gender: "female");

        // Print each students name
        System.out.println("Name of student1: " + student1.getName());
        System.out.println("Name of student2: " + student2.getName());
        System.out.println("Name of student3: " + student3.getName());

        student1.setName("john");
        System.out.println("Updated name of student1: " + student1.getName());
    }
}
```



```
class StudentTest {

    public static void main(String[] args) {
        int[] studentIds = new int[] {1001, 1002, 1003};

        // Creating first student object and setting its state
        Student student1 = new Student(studentIds[0], name: "joan", gender: "male");

        // Creating second student object and setting its state
        Student student2 = new Student(studentIds[1], name: "raj", gender: "male");

        // Creating third student object and setting its state
        Student student3 = new Student(studentIds[2], name: "anita", gender: "female");

        // Print each students name
        System.out.println("Name of student1: " + student1.getName());
        System.out.println("Name of student2: " + student2.getName());
        System.out.println("Name of student3: " + student3.getName());

        student1.setName("john");
        System.out.println("Updated name of student1: " + student1.getName());
```

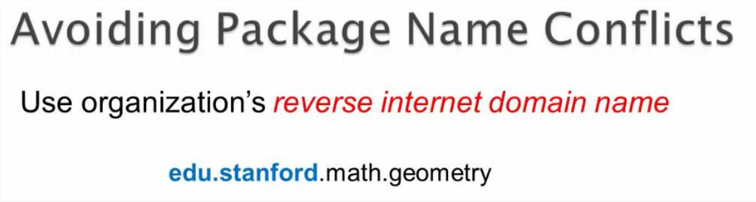
```
"C:\Program Files\Java\jdk1.8.0_221\bin\java" ...
Name of student1: joan
Name of student2: raj
Name of student3: anita
Updated name of student1: john

Process finished with exit code 0
```

Какво научихте до сега:



Accessing Classes

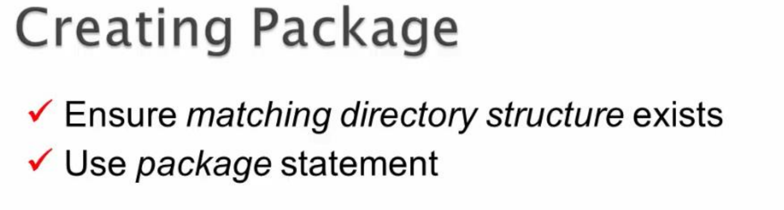
- import
  - Explicit import preferred over * import
  - *Doesn't* make classes *bigger*
  - *Doesn't* affect runtime performance
  - Saves from typing fully-qualified class names
  - **java.lang** is imported by default



Avoiding Package Name Conflicts

Use organization's *reverse internet domain name*

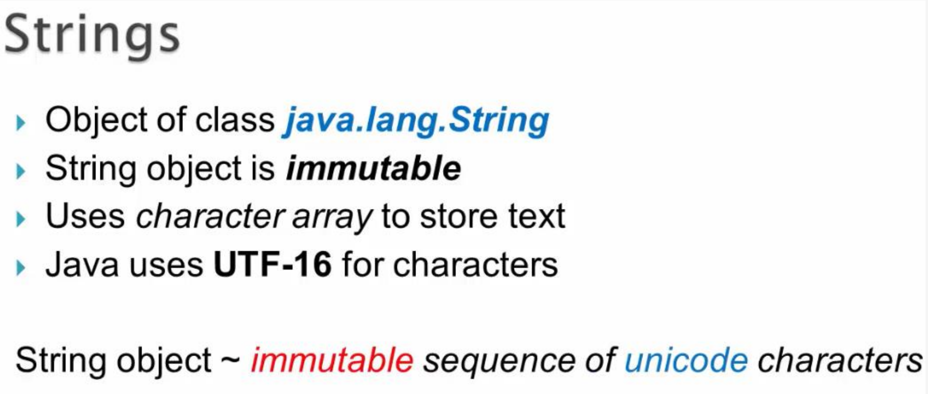
edu.stanford.math.geometry



Creating Package

✓ Ensure *matching directory structure* exists

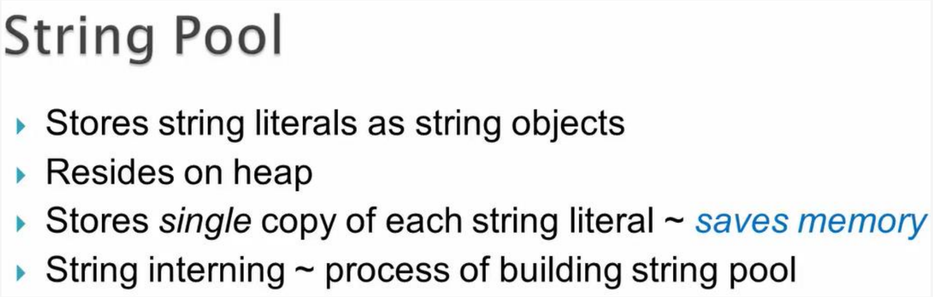
✓ Use *package* statement



Strings

- Object of class ***java.lang.String***
- String object is ***immutable***
- Uses *character array* to store text
- Java uses **UTF-16** for characters

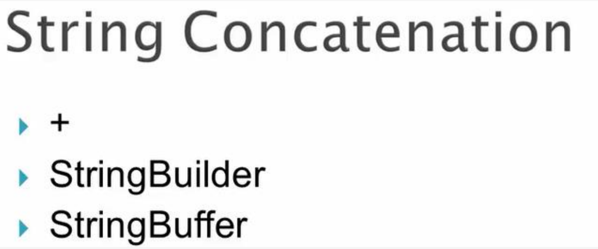
String object ~ *immutable sequence of unicode characters*



String Pool

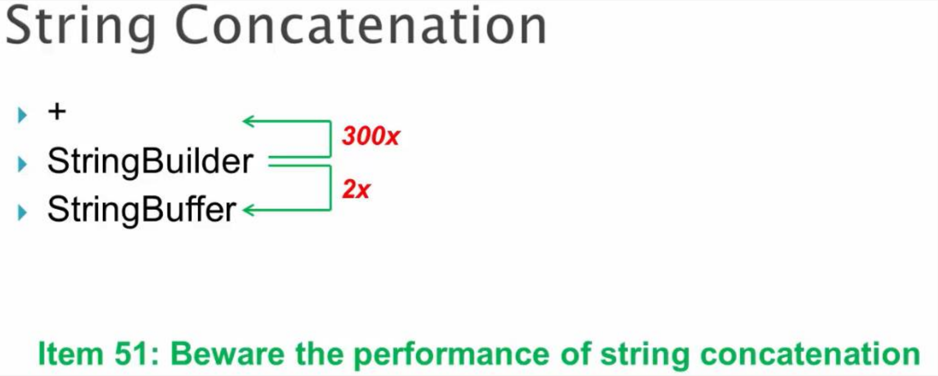
- Stores string literals as string objects
- Resides on heap
- Stores *single* copy of each string literal ~ *saves memory*
- String interning ~ process of building string pool

Какво научихте до сега:



String Concatenation

- +
- StringBuilder
- StringBuffer



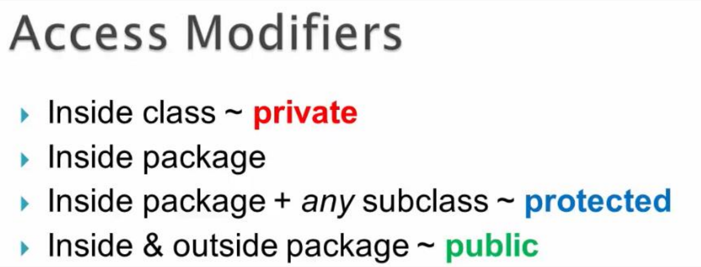
String Concatenation

- +
- StringBuilder
- StringBuffer

300x

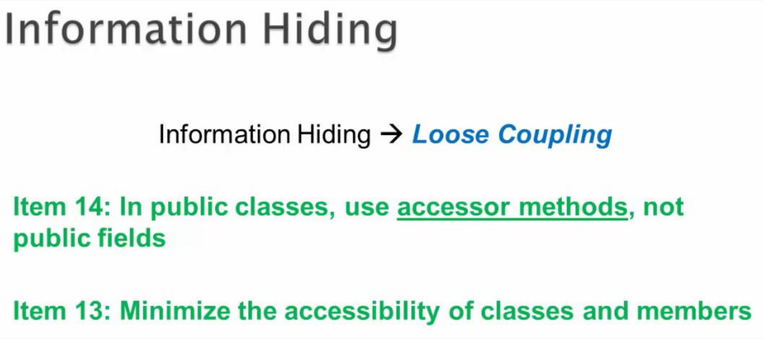
2x

Item 51: Beware the performance of string concatenation



Access Modifiers

- Inside class ~ **private**
- Inside package
- Inside package + *any* subclass ~ **protected**
- Inside & outside package ~ **public**



Information Hiding

Information Hiding → ***Loose Coupling***

Item 14: In public classes, use accessor methods, not public fields

Item 13: Minimize the accessibility of classes and members