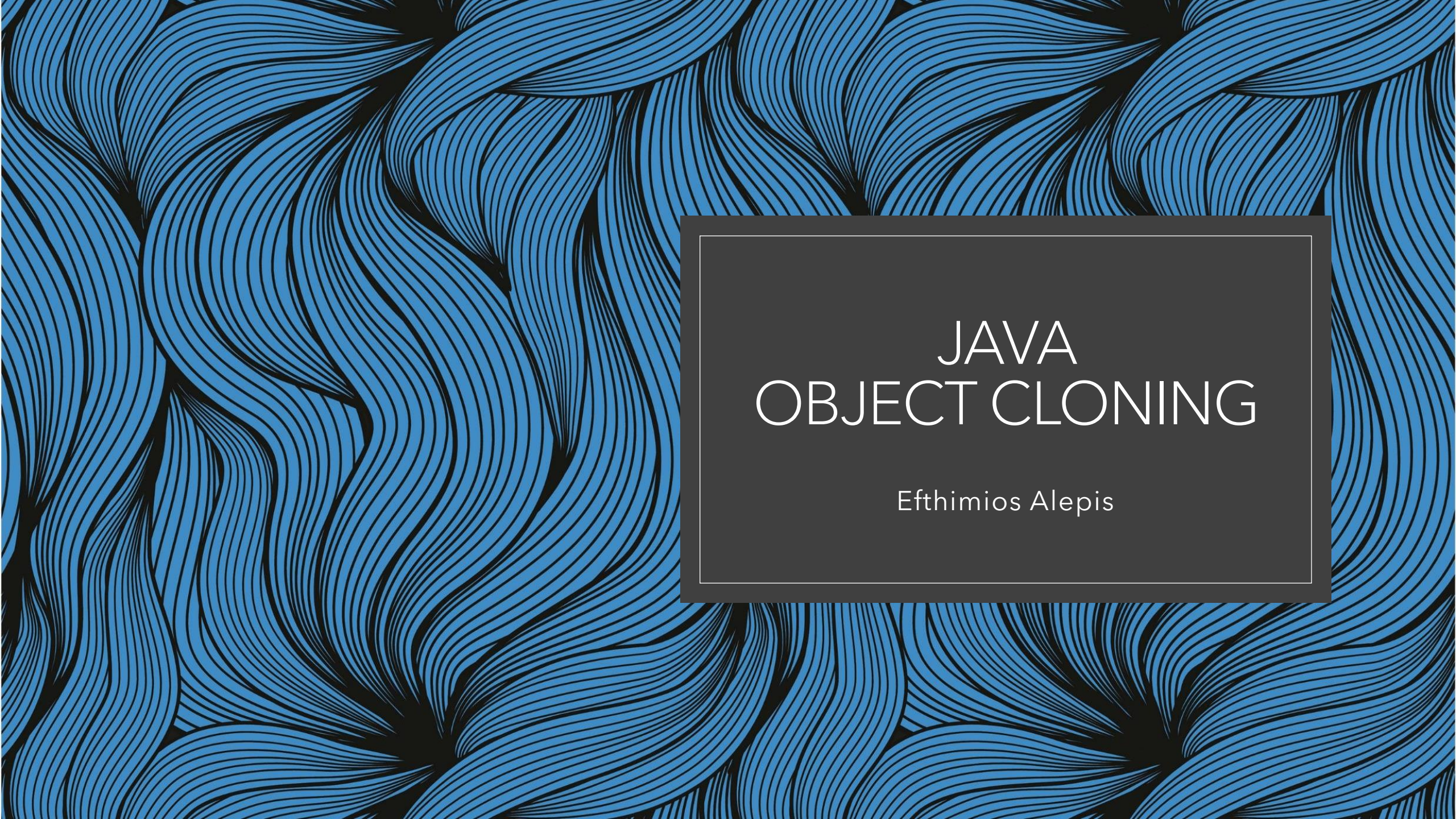


JAVA OBJECT CLONING

Efthimios Alepis

What is Cloning

- Creating an identical copy of the original Object
- In Java by default tis is implemented by copying the all the fields of the object, one-by-one

Default cloning by the JVM

- If the class contains primitive data type members then a completely new copy of them will be created and the reference to the new object copy will be returned
- If the class contains members of any class type then only the object references to those members are copied and hence the member references in both the original object as well as the cloned object refer to the same object
- So primitive types are represented by new copies, while reference types are copied as references

Cloneable Interface

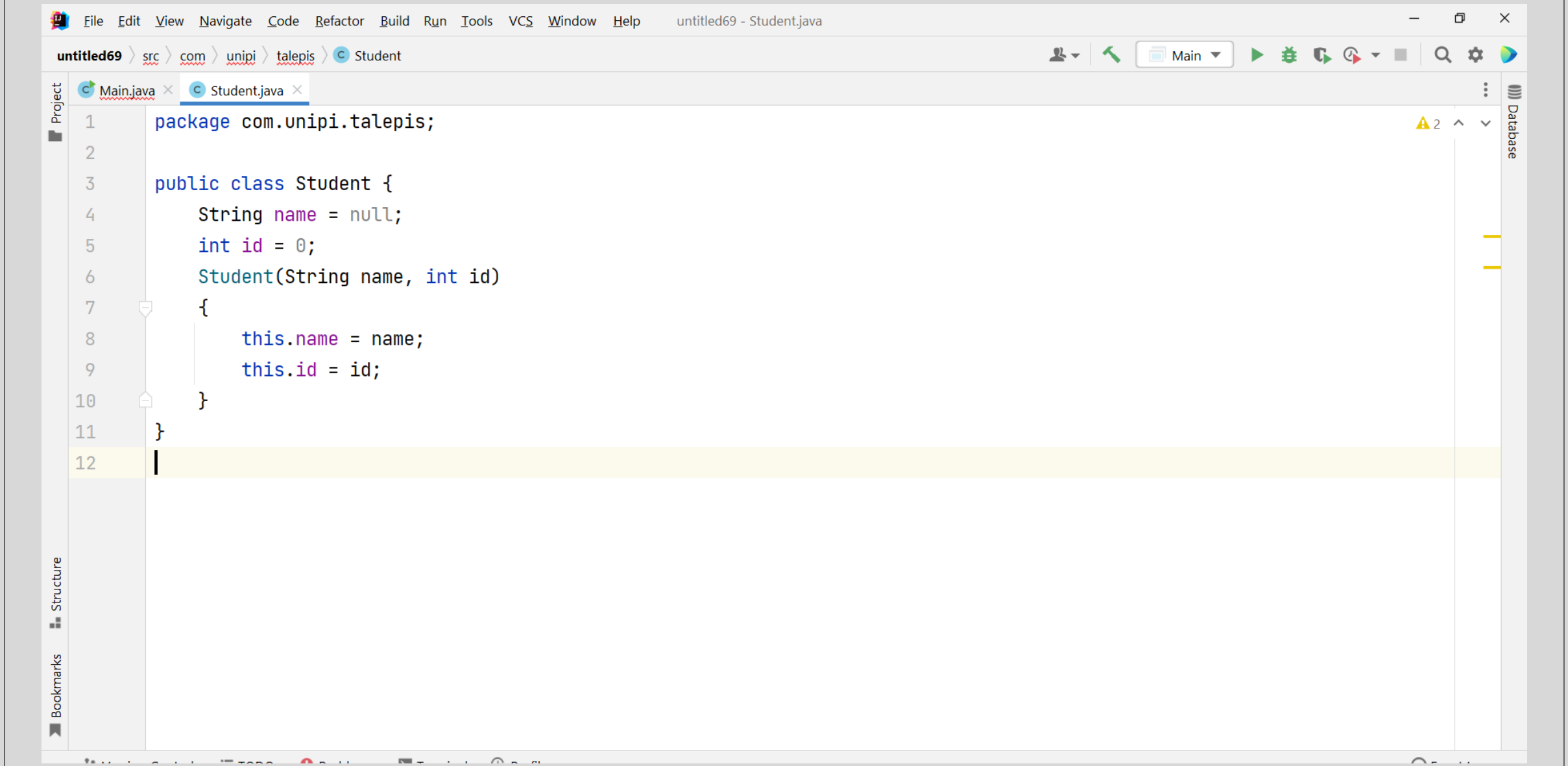
- It is a marker Interface (Important: It does not contain an abstract method…)
- A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class.
- Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception CloneNotSupportedException being thrown.
- By convention, classes that implement this interface should override Object.clone (which is protected) with a public method. See Object.clone() for details on overriding this method.
- Note that this interface does not contain the clone method. Therefore, it is not possible to clone an object merely by virtue of the fact that it implements this interface. Even if the clone method is invoked reflectively, there is no guarantee that it will succeed.

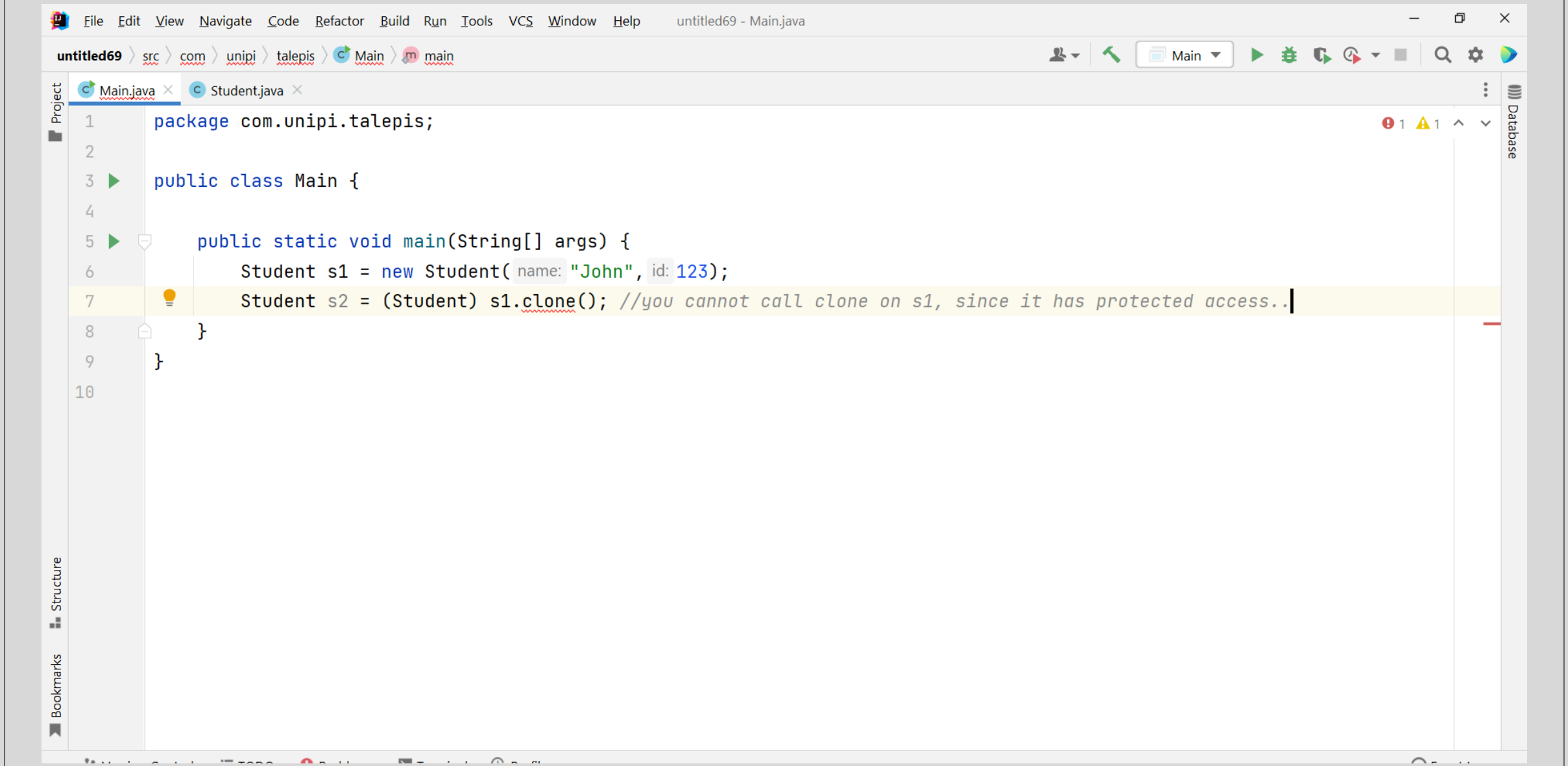
Use Cloning in Java

- You must implement Cloneable interface.
- You must override clone() method from Object class.
- You need both of the above!..
- From Java docs:

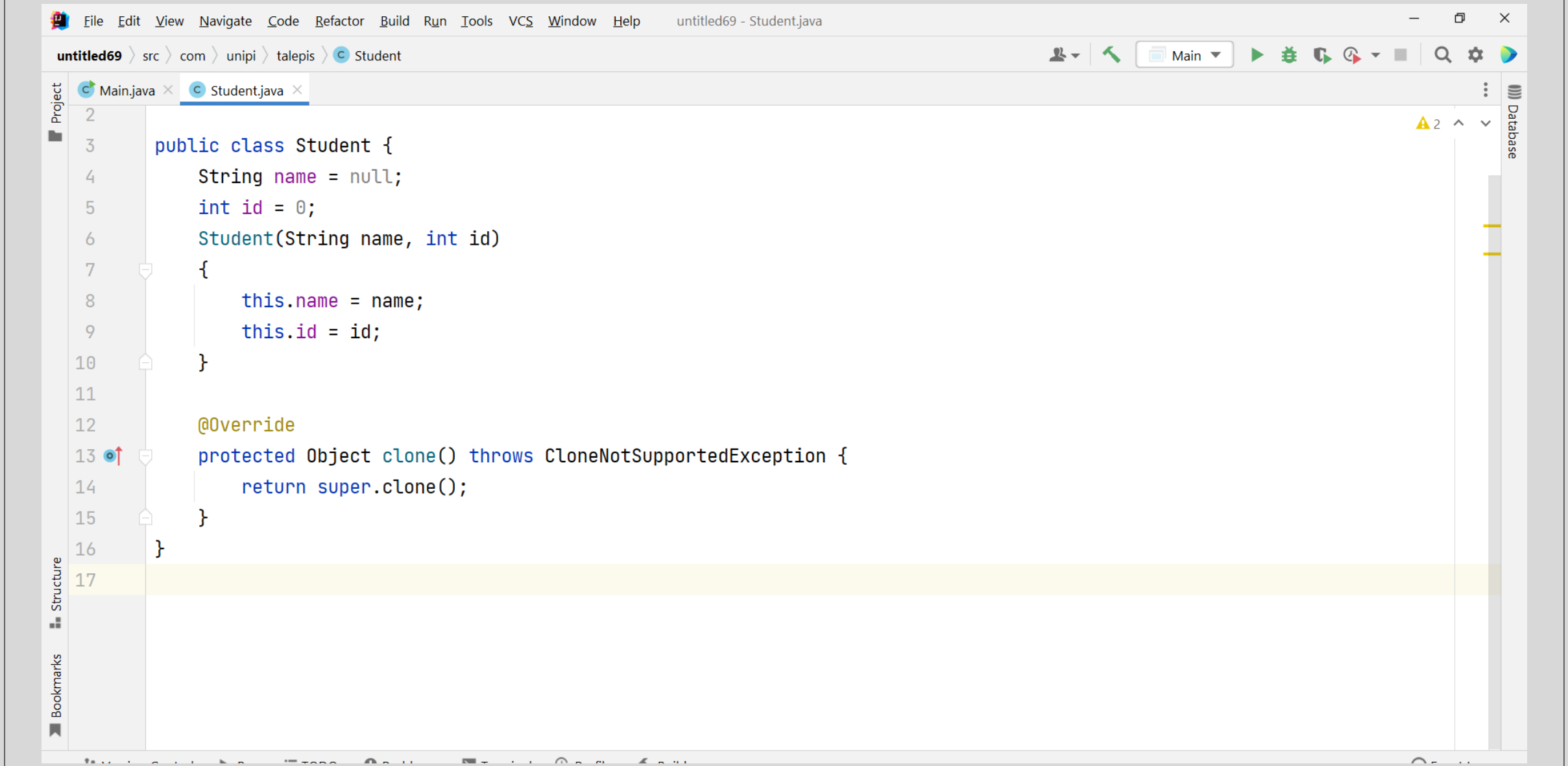
```
1) x.clone() != x will be true
2) x.clone().getClass() == x.getClass() will be true, but these are not
absolute requirements.
3) x.clone().equals(x) will be true, this is not an absolute requirement.
```

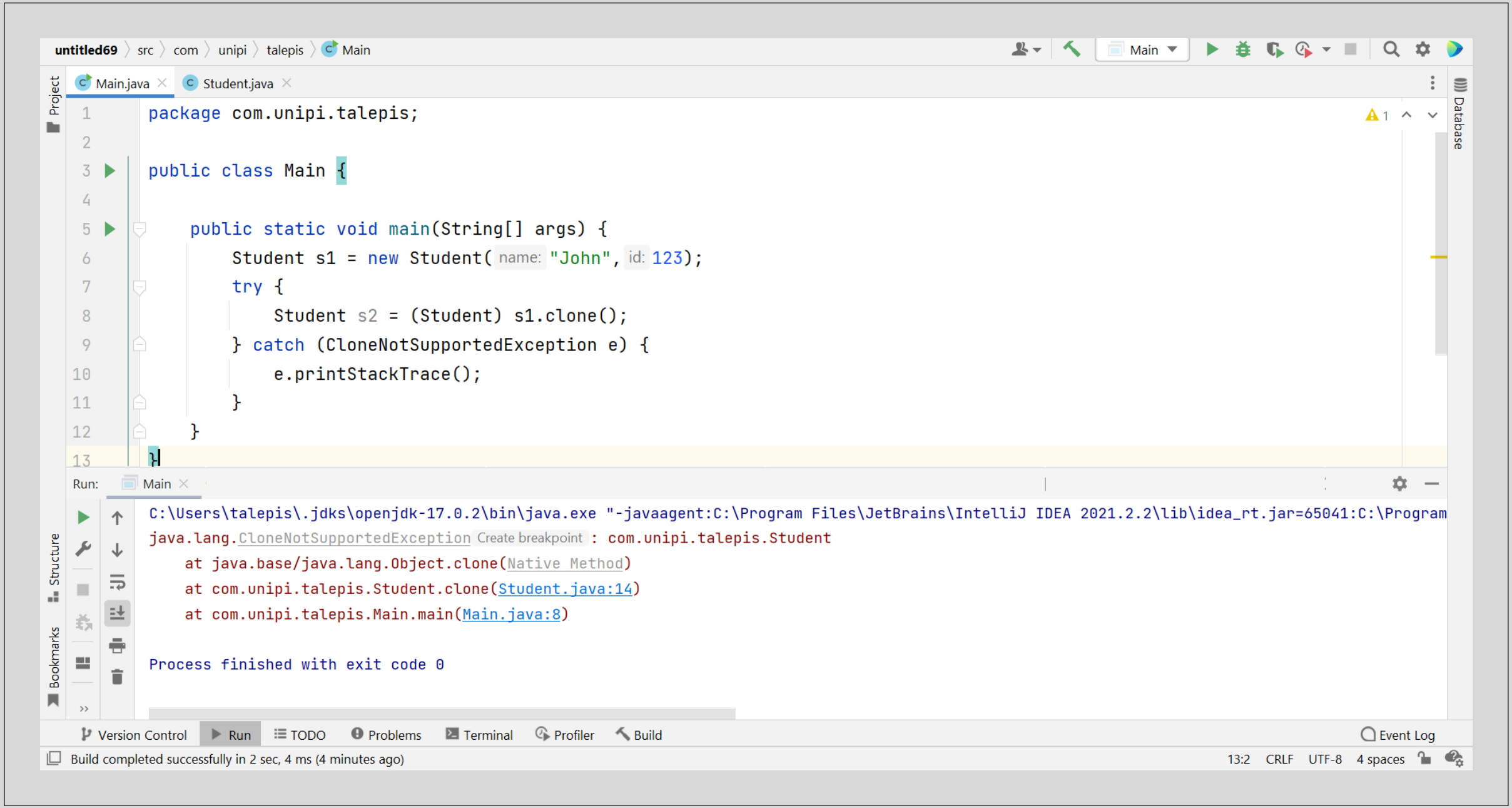
EXAMPLE


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
untitled69 - Student.java
untitled69 〉 src 〉 com 〉 unipi 〉 talepis 〉 Student
Main
Main.java
Student.java
Project
Database
Structure
Bookmarks
package com.unipi.talepis;
public class Student {
String name = null;
int id = 0;
Student(String name, int id)
{
this.name = name;
this.id = id;
}
}

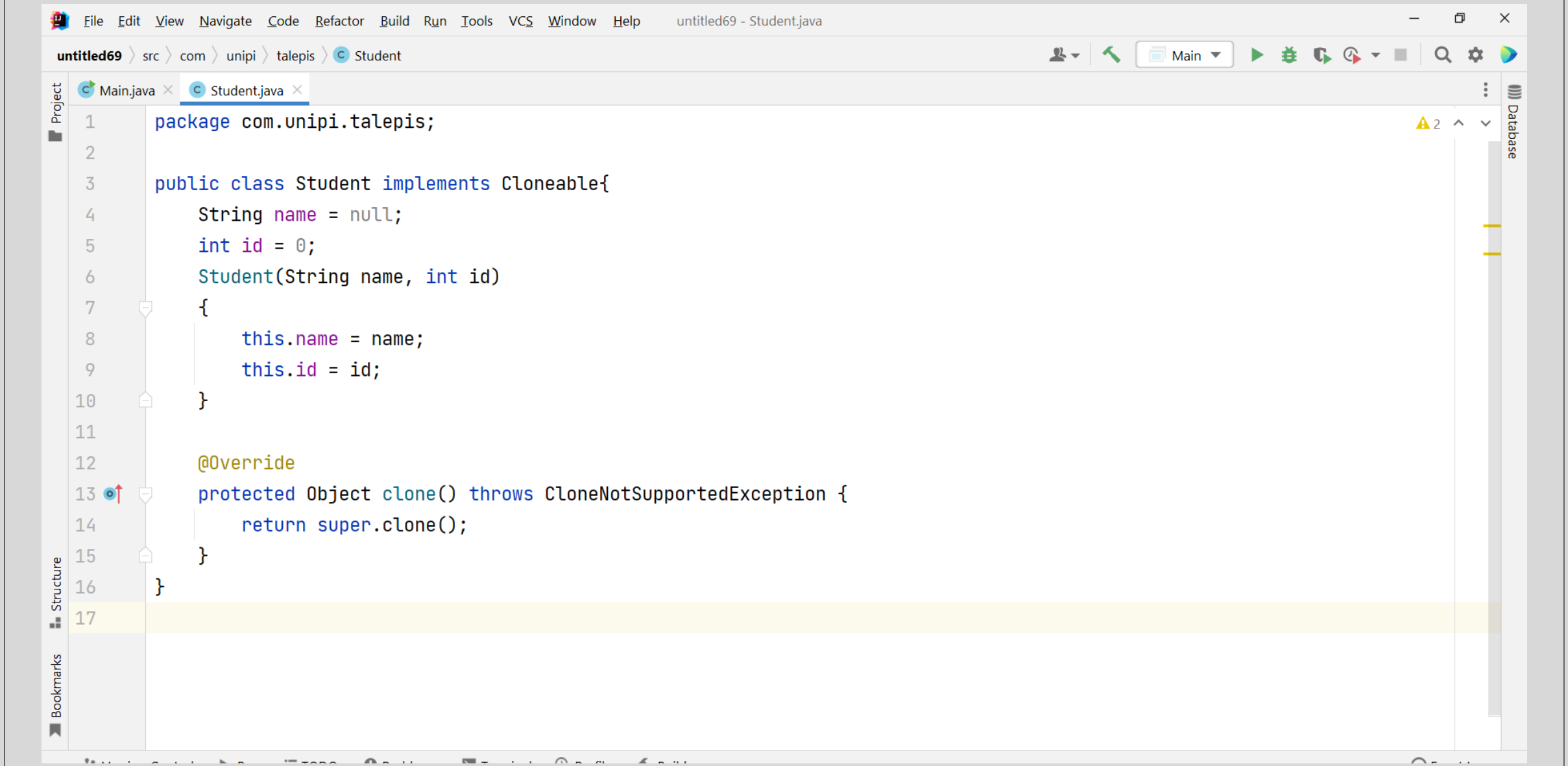

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
untitled69 - Main.java
untitled69 src com unipi talepis Main main
Main
Main.java
Student.java
1 package com.unipi.talepis;
2
3 public class Main {
4
5 public static void main(String[] args) {
6 Student s1 = new Student(name: "John", id: 123);
7 Student s2 = (Student) s1.clone(); //you cannot call clone on s1, since it has protected access..
8 }
9 }
10
Project
Structure
Bookmarks
Database

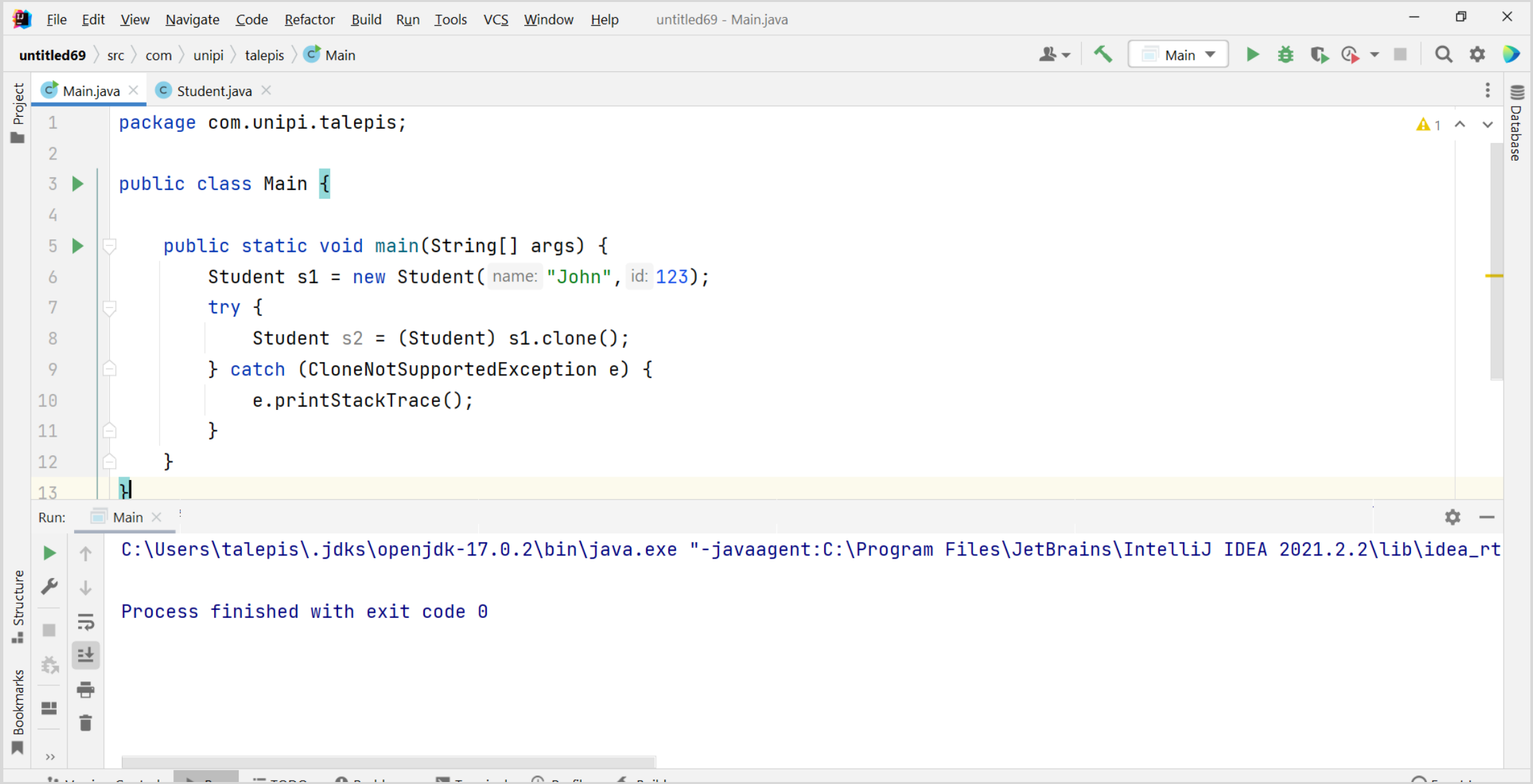
Override clone()


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
untitled69 - Student.java
untitled69 〉 src 〉 com 〉 unipi 〉 talepis 〉 Student
Main
Main.java
Student.java
Project
Database
Structure
Bookmarks
2
public class Student {
String name = null;
int id = 0;
Student(String name, int id)
{
this.name = name;
this.id = id;
}
@Override
protected Object clone() throws CloneNotSupportedException {
return super.clone();
}
}


untitled69 src com unipi talepis Main
Main
Main.java
Student.java
1 package com.unipi.talepis;
2
3 public class Main {
4
5 public static void main(String[] args) {
6 Student s1 = new Student(name: "John", id: 123);
7 try {
8 Student s2 = (Student) s1.clone();
9 } catch (CloneNotSupportedException e) {
10 e.printStackTrace();
11 }
12 }
13 }
Run: Main
C:\Users\talepis\.jdks\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.2\lib\idea_rt.jar=65041:C:\Program
java.lang.CloneNotSupportedException Create breakpoint : com.unipi.talepis.Student
at java.base/java.lang.Object.clone(Native Method)
at com.unipi.talepis.Student.clone(Student.java:14)
at com.unipi.talepis.Main.main(Main.java:8)
Process finished with exit code 0
Project
Database
Structure
Bookmarks
Version Control
Run
TODO
Problems
Terminal
Profiler
Build
Event Log
Build completed successfully in 2 sec, 4 ms (4 minutes ago)
13:2 CRLF UTF-8 4 spaces

Override clone(), implement Cloneable


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
untitled69 - Student.java
untitled69 〉 src 〉 com 〉 unipi 〉 talepis 〉 Student
Main
Main.java
Student.java
Project
Database
Structure
Bookmarks
1 package com.unipi.talepis;
2
3 public class Student implements Cloneable{
4 String name = null;
5 int id = 0;
6 Student(String name, int id)
7 {
8 this.name = name;
9 this.id = id;
10 }
11
12 @Override
13 protected Object clone() throws CloneNotSupportedException {
14 return super.clone();
15 }
16 }
17
2


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
untitled69 - Main.java
untitled69 〉 src 〉 com 〉 unipi 〉 talepis 〉 Main
Main
Main.java
Student.java
package com.unipi.talepis;
public class Main {
public static void main(String[] args) {
Student s1 = new Student(name: "John", id: 123);
try {
Student s2 = (Student) s1.clone();
} catch (CloneNotSupportedException e) {
e.printStackTrace();
}
}
}
Run:
Main
C:\Users\talepis\.jdks\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.2\lib\idea_rt
Process finished with exit code 0
Project
Database
Structure
Bookmarks

Cloning and Copying

- Both mean practically the same thing in terms of result
- Cloning needs the clone() method to be used
- Copying could include the clone() method, but it can be achieved with other ways as well
- There is a very big difference regarding the way we Copy, namely:
  - Deep Copying and
  - Shallow Copying

Deep Copy Vs Shallow Copy

- Shallow clone is the “default” implementation in Java. While overriding the clone() method, if you are not cloning all the object types (not primitives), then you are making a shallow copy. Shallow copy can be achieved without using the clone() method. With shallow copy you don’t create new variables for the reference type fields
- Deep copying is the desired behavior in most of the cases. In the deep copy, we create a real “clone” which is independent of original object and making changes in the cloned object should not affect the original object. Nevertheless, deep copying is much more difficult to be achieved and needs special care.
- With deep copy, each mutable object in the object graph is recursively copied

Deep copy basics

Primitives and immutable objects: No need to copy them with special care

Mutable objects: each mutable object in the object graph should be recursively copied

There exist different approaches in deep copying objects:

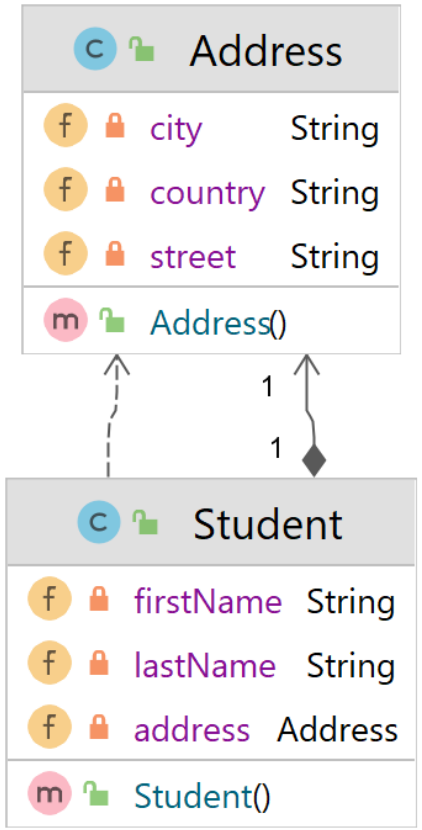
Deep copy with copy constructor	Clone with deep copy implementation	Deep copy with serialization	Deep copy with external libraries (e.g. Apache commons)

Deep copy with copy Constructor

Copy constructors are special constructors in a class that take an argument of its own class type

Passing an instance of a class to the copy constructor will return a new instance of class with values copied from the argument instance

MODEL



*Please note that String is an Immutable class

```java
package com.unipi.talepis.deepcopying.copyconstructor;

public class Address {
    private String street;
    private String city;
    private String country;
    // standard getters and setters

    public Address(Address a) {
        this.street = a.street;
        this.city = a.city;
        this.country = a.country;
    }
}
```

```java
package com.unipi.talepis.deepcopying.copyconstructor;

public class Student {
    private String firstName;
    private String lastName;
    private Address address;
    // standard getters and setters

    public Student(Student s) {
        this.firstName = s.firstName;
        this.lastName = s.lastName;
        this.address = new Address(s.address);
    }
}
```

1.COPY CONSTRUCTOR

Notes about copy constructor

- You should also provide other ways of “first “ instantiation
- The copy constructor will be used for “cloning” purposes
- Be careful about mutable fields inside the copied object. If present, you should also take care for their deep copying (possible through copy constructor as well)

Using default cloning approach

- We will use the previous model
- We will initially try the basic cloning approach

```java
package com.unipi.talepis.deepcopying.copyconstructor;

public class Student implements Cloneable{
    private String firstName;
    private String lastName;
    private Address address;

    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName = firstName; }
    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }
    public Address getAddress() { return address; }
    public void setAddress(Address address) { this.address = address; }

    public Student(String firstName, String lastName, Address address) {...}
    public Student(Student s) {...}

    @Override
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

```java
package com.unipi.talepis;

import com.unipi.talepis.deepcopying.copyconstructor.Address;
import com.unipi.talepis.deepcopying.copyconstructor.Student;

public class Main {

    public static void main(String[] args) {
        Address ad1 = new Address( street: "Karaoli", city: "Piraeus", country: "Greece");
        Student std1 = new Student( firstName: "John", lastName: "Pap",ad1);
        //Clone process
        try {
            Student clone = (Student) std1.clone();
            //Now let's test it
            clone.getAddress().setCity("Athens");
            System.out.println(std1.getAddress().getCity());//prints Athens
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }
}
```

Default cloning approach results

- As we see from the previous example, though cloning seems to work, only shallow copy is achieved
- The “address” field which is clearly mutable, is not deep copied
- As a result, changing values in the cloned object affects the initial object!..

2. Using custom cloning approach

- We will use the previous model
- We will do a deep copy
- Each mutable field will be handled separately

```java
package com.unipi.talepis.deepcopying.copyconstructor;

public class Address implements Cloneable{
    private String street;
    private String city;
    private String country;
    // standard getters and setters

    public String getStreet() { return street; }
    public void setStreet(String street) { this.street = street; }
    public String getCity() { return city; }
    public void setCity(String city) { this.city = city; }
    public String getCountry() { return country; }
    public void setCountry(String country) { this.country = country; }

    public Address(String street, String city, String country) {...}
    public Address(Address a) {...}

    @Override
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

```java
package com.unipi.talepis.deepcopying.copyconstructor;

public class Student implements Cloneable{
    private String firstName;
    private String lastName;
    private Address address;

    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName = firstName; }
    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }
    public Address getAddress() { return address; }
    public void setAddress(Address address) { this.address = address; }

    public Student(String firstName, String lastName, Address address) {...}
    public Student(Student s) {...}

    @Override
    public Object clone() throws CloneNotSupportedException {
        return new Student(getFirstName(),getLastName(),(Address) getAddress().clone());
    }
}
```

```java
package com.unipi.talepis;

import com.unipi.talepis.deepcopying.copyconstructor.Address;
import com.unipi.talepis.deepcopying.copyconstructor.Student;

public class Main {

    public static void main(String[] args) {
        Address ad1 = new Address( street: "Karaoli", city: "Piraeus", country: "Greece");
        Student std1 = new Student( firstName: "John", lastName: "Pap",ad1);
        //Clone process
        try {
            Student clone = (Student) std1.clone();
            //Now let's test it
            clone.getAddress().setCity("Athens");
            System.out.println(std1.getAddress().getCity());//Now it's ok, prints Piraeus
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }
}
```

3. Using a serialization approach

- We will use the previous model