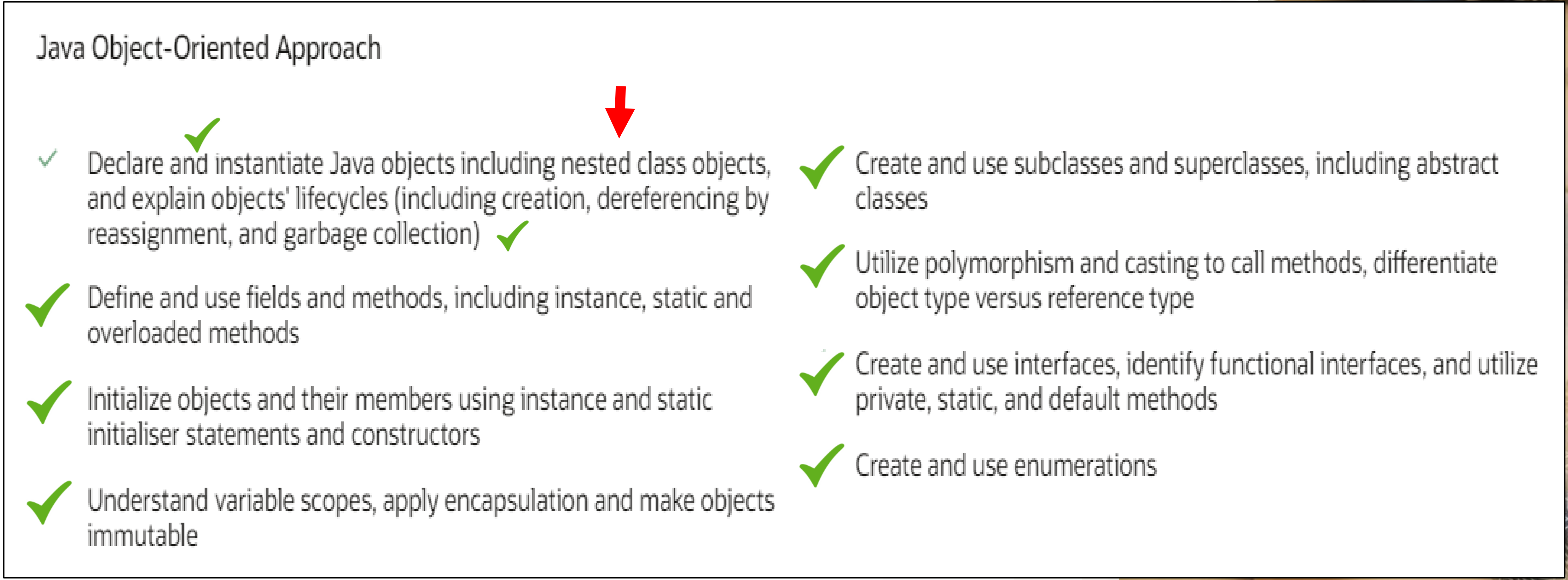
Java Object Oriented Approach

Nested classes

Java Object-Oriented Approach



Java Object-Oriented Approach

- ✓ Declare and instantiate Java objects including nested class objects, and explain objects' lifecycles (including creation, dereferencing by reassignment, and garbage collection) ✓
- ✓ Define and use fields and methods, including instance, static and overloaded methods
- ✓ Initialize objects and their members using instance and static initialiser statements and constructors
- ✓ Understand variable scopes, apply encapsulation and make objects immutable
- ✓ Create and use subclasses and superclasses, including abstract classes
- ✓ Utilize polymorphism and casting to call methods, differentiate object type versus reference type
- ✓ Create and use interfaces, identify functional interfaces, and utilize private, static, and default methods
- ✓ Create and use enumerations

Nested Classes

- A nested class is a class that is defined within another class.

- When are they used:
  - ➢ where you require a class that will only be used in one place
  - ➢to encapsulate helper classes to their containing classes

- There are four different types:
  1. Inner class (non-static class, member scope)
  2. Static nested class (static class, member scope)
  3. Local class (local to a method i.e. method scope)
  4. Anonymous inner class (a special local class which has no name)

Inner Classes

- An inner class is non-static and is defined at the same level of scope as methods and constructors.

- Must be associated with an instance of the outer class.

- The inner class has access to the *private* members of the outer class.

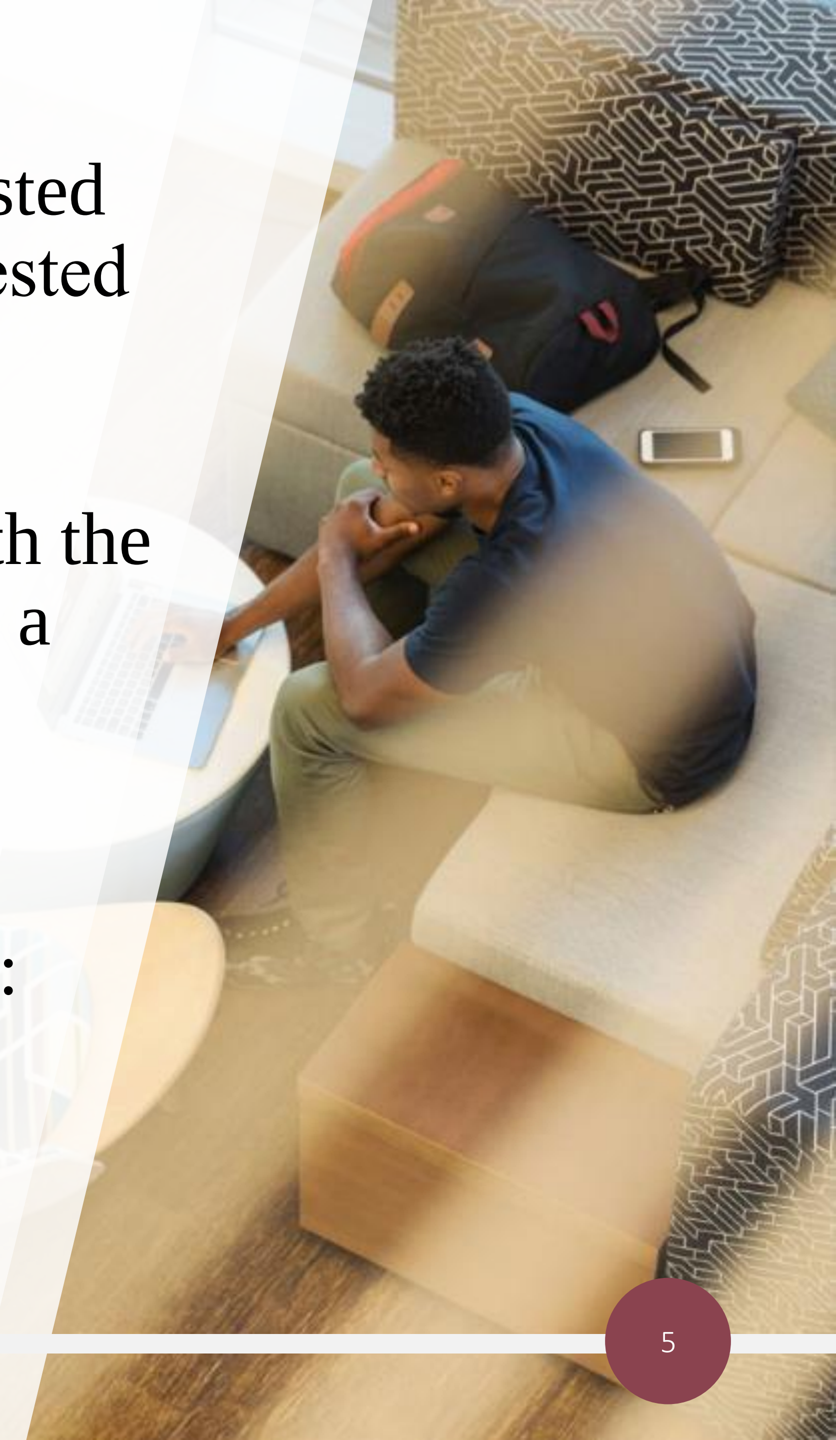
Static Nested Classes

- While an “inner class” refers to a non-static inner/nested class, a “static nested class” refers to a static inner/nested class.

- While an inner class enjoys a special relationship with the outer class (i.e. the instances of the two classes share a relationship), a static nested class does not.

- A static nested class is simply a class that is, at the member level, a static member of the enclosing class:

```
class Outer{
      static class Nested{}
}
```

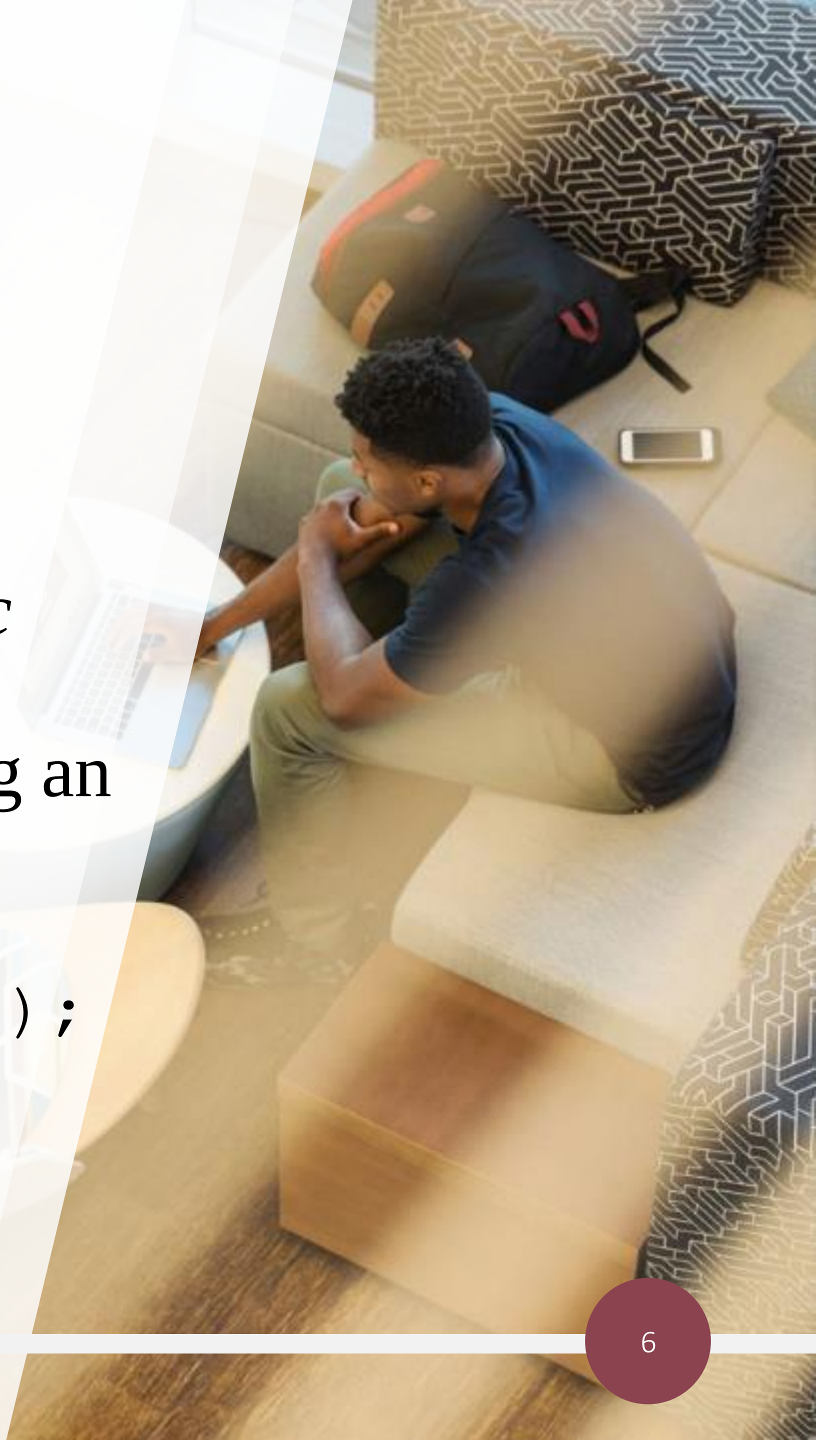


Static Nested Classes

-
  ```
  class Outer{
          static class Nested{}
  }
  ```

- The *static* modifier says that the nested class is a *static* member of the outer class. This means that it can be accessed, as with other *static* members, without having an instance of the outer class:

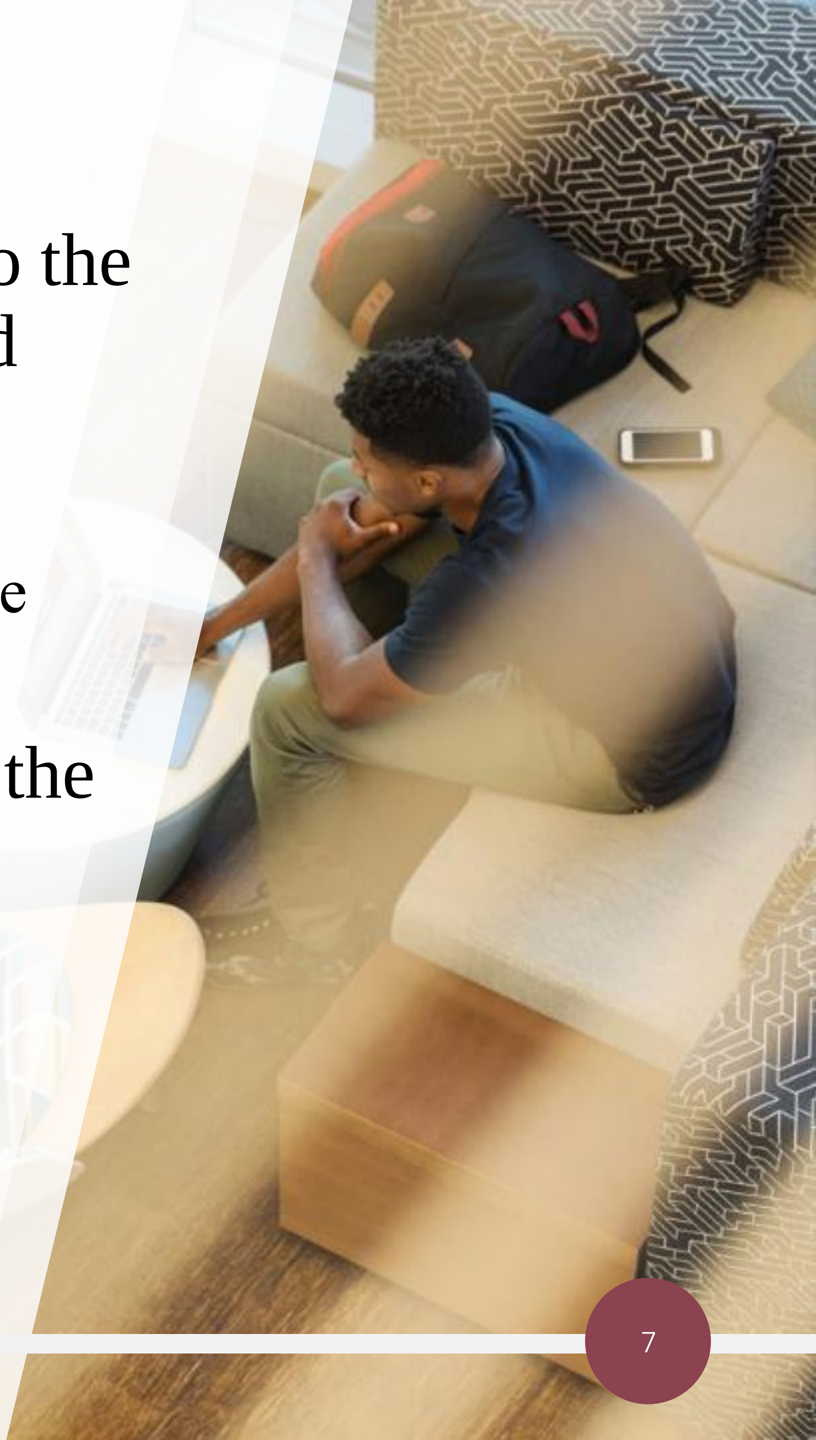
  ```
  Outer.Nested nested = new Outer.Nested();
  ```



Static Nested Classes

- Just as a static method does not have (direct) access to the instance variables/methods of the class, a static nested class does not have (direct) access to the instance variables/methods of the outer class.
  - “direct” = access can be gained via an outer class reference

- The outer class can refer to the fields and methods of the *static* nested class.



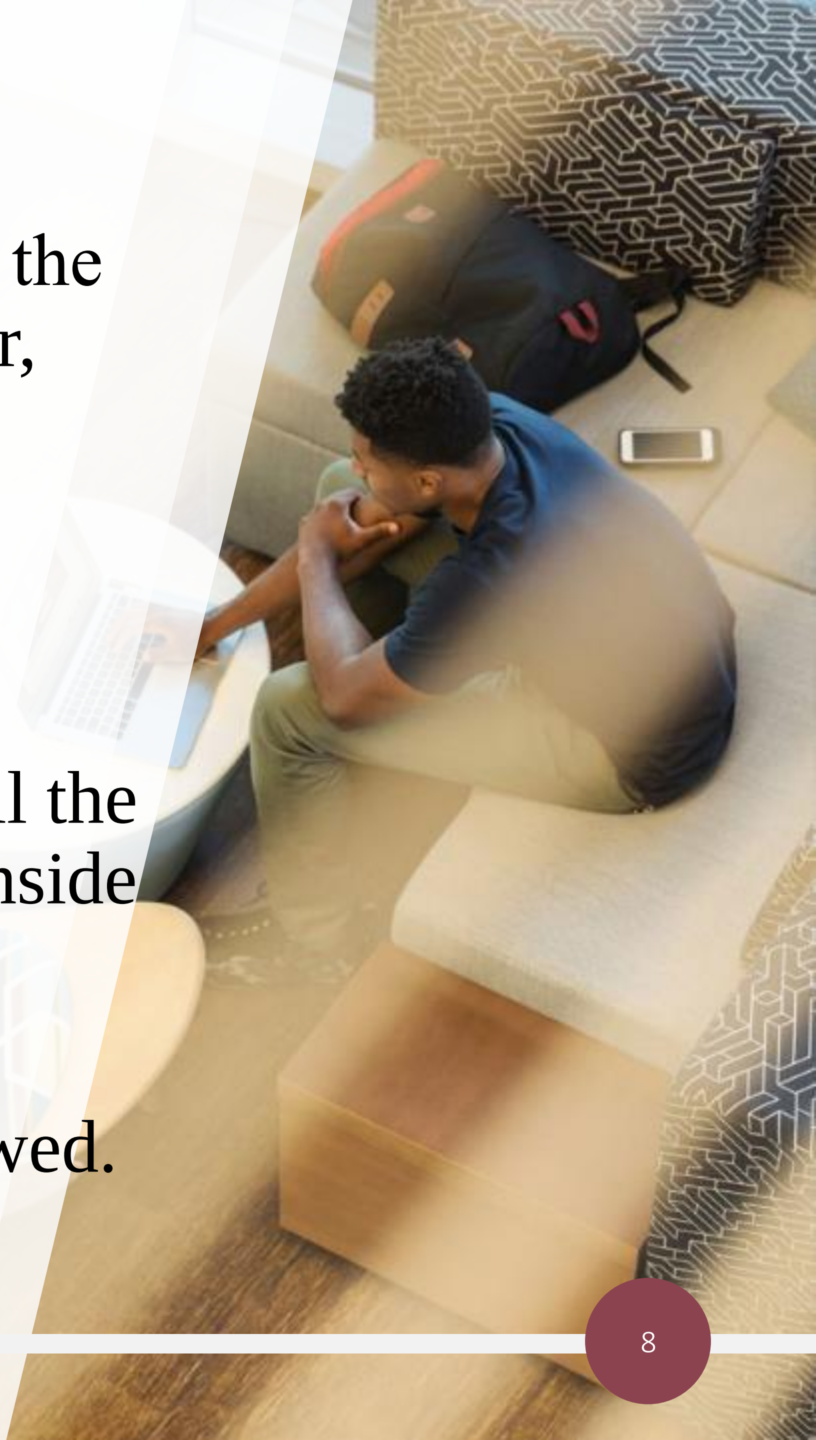
Local Inner Classes

- A local inner class is a “method-local” inner class i.e. the class definition occurs inside a method (or constructor, initialisation block).

- Instantiate the class after you have defined it.

- As with inner classes, a local inner class can access all the fields and methods of the outer class (when defined inside an instance method).

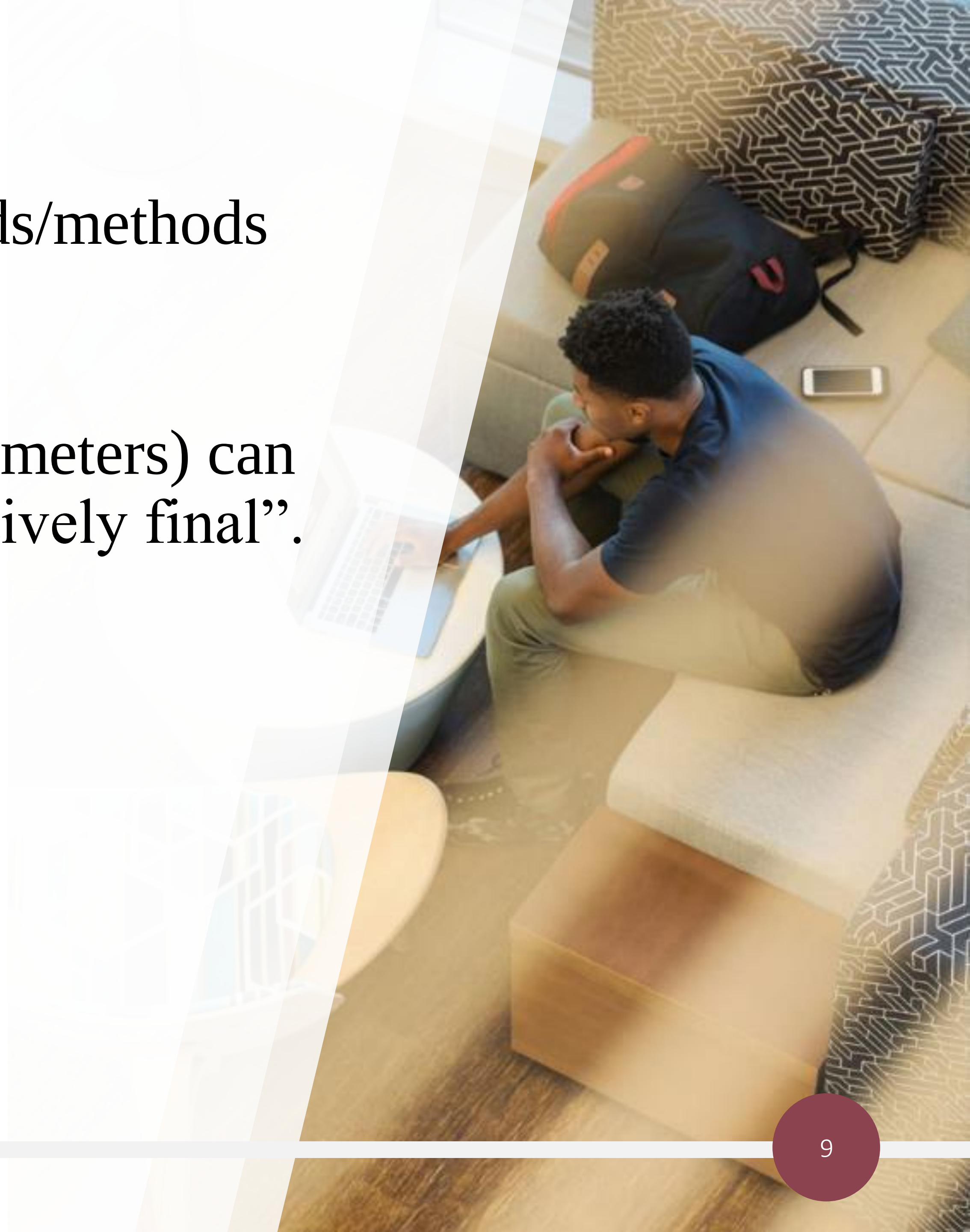
- As with local variables, access modifiers are not allowed.



Local Inner Classes

- They cannot be *static* or contain *static* fields/methods (except for *static final* fields).

- The local variables (including method parameters) can **only** be accessed if they are *final* or “effectively final”.



Anonymous Inner Classes

- A specialised type of local inner class which has no name.

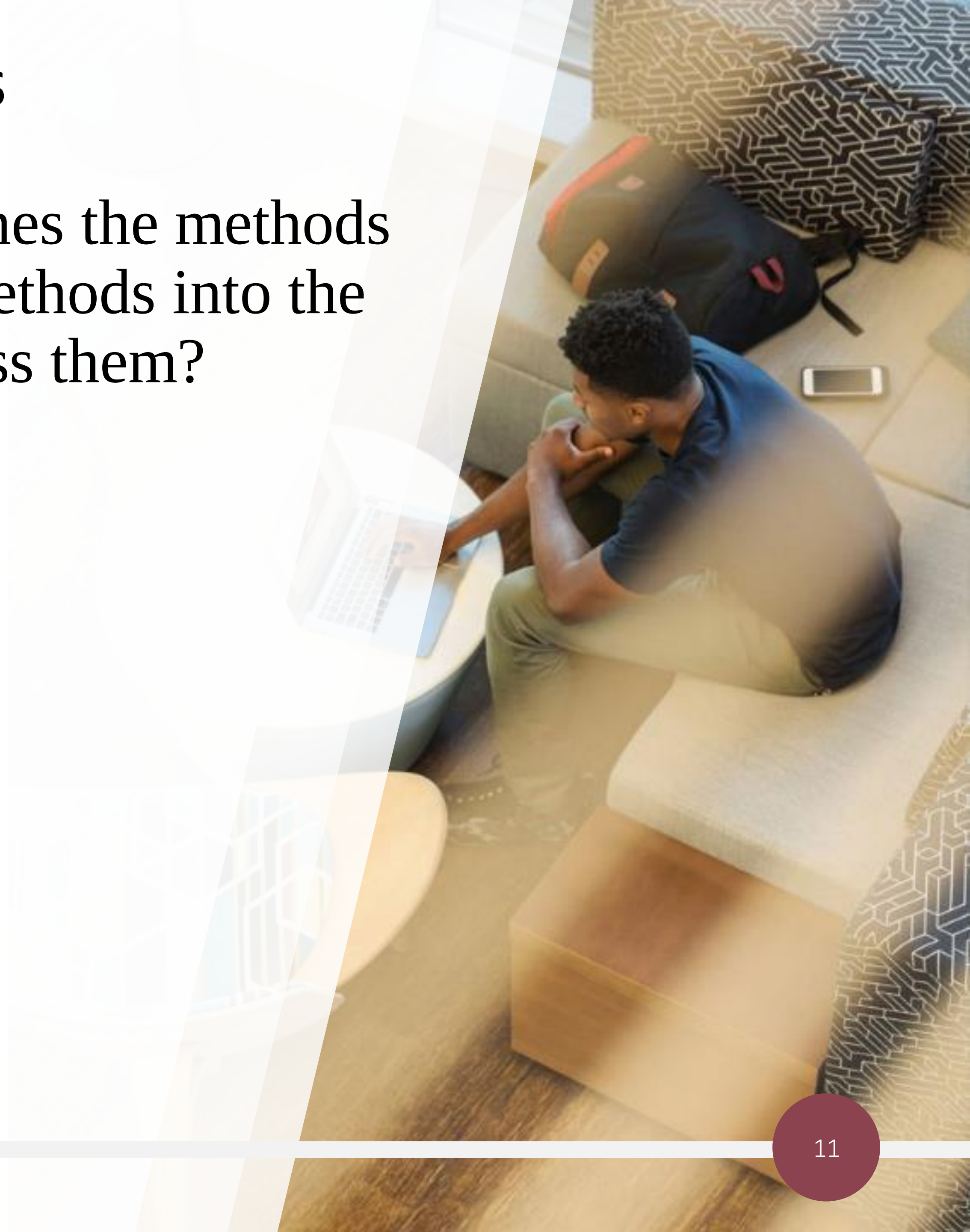
- They are typically local:
  - ➢ defined within a method
  - ➢within an argument to a method

- You can define them right where you need them.

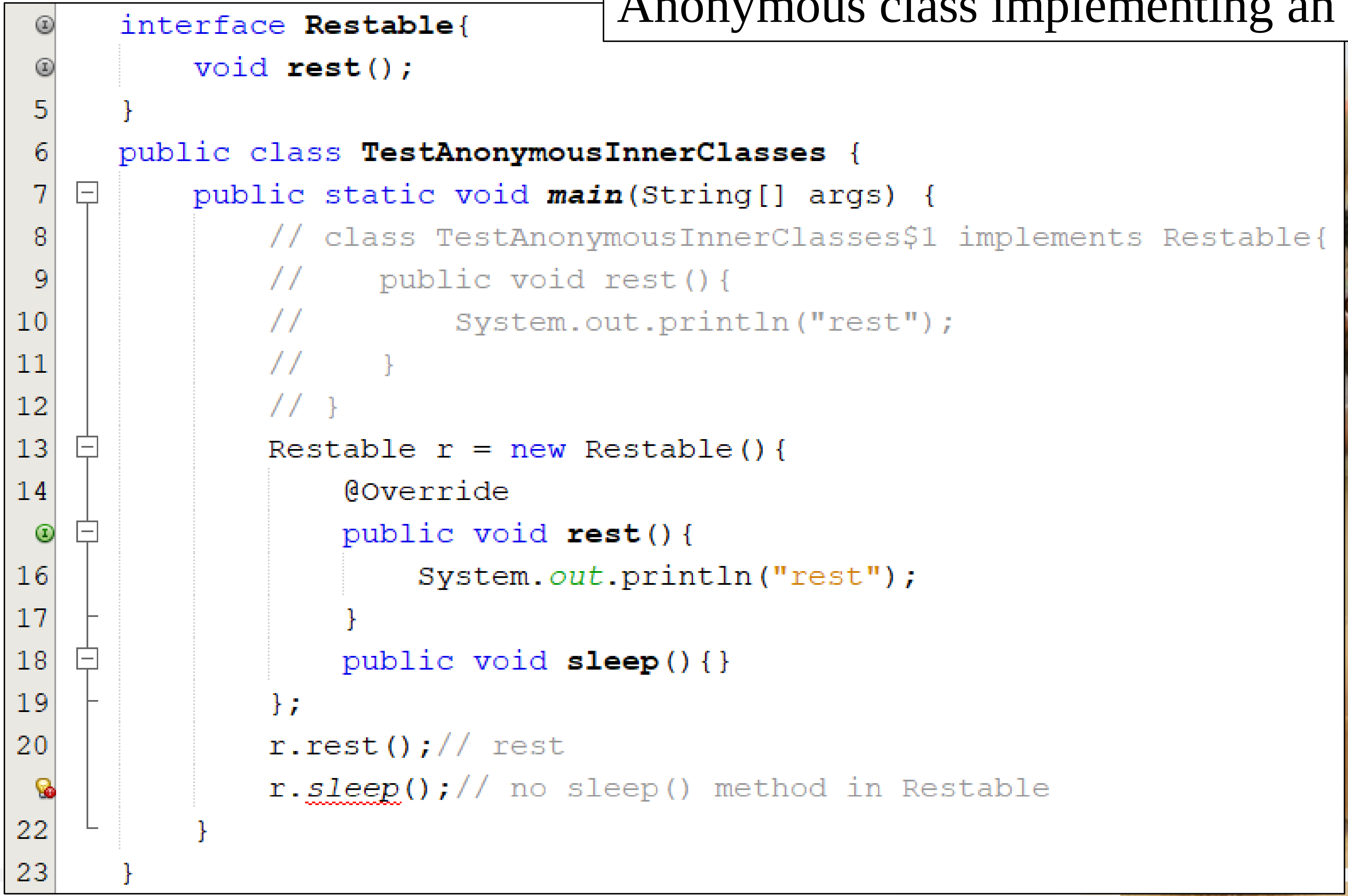
- They either extend a class or implement a single interface.

Anonymous Inner Classes

- Remember that the reference type determines the methods you can access i.e. if you introduce new methods into the anonymous inner class, how will you access them?

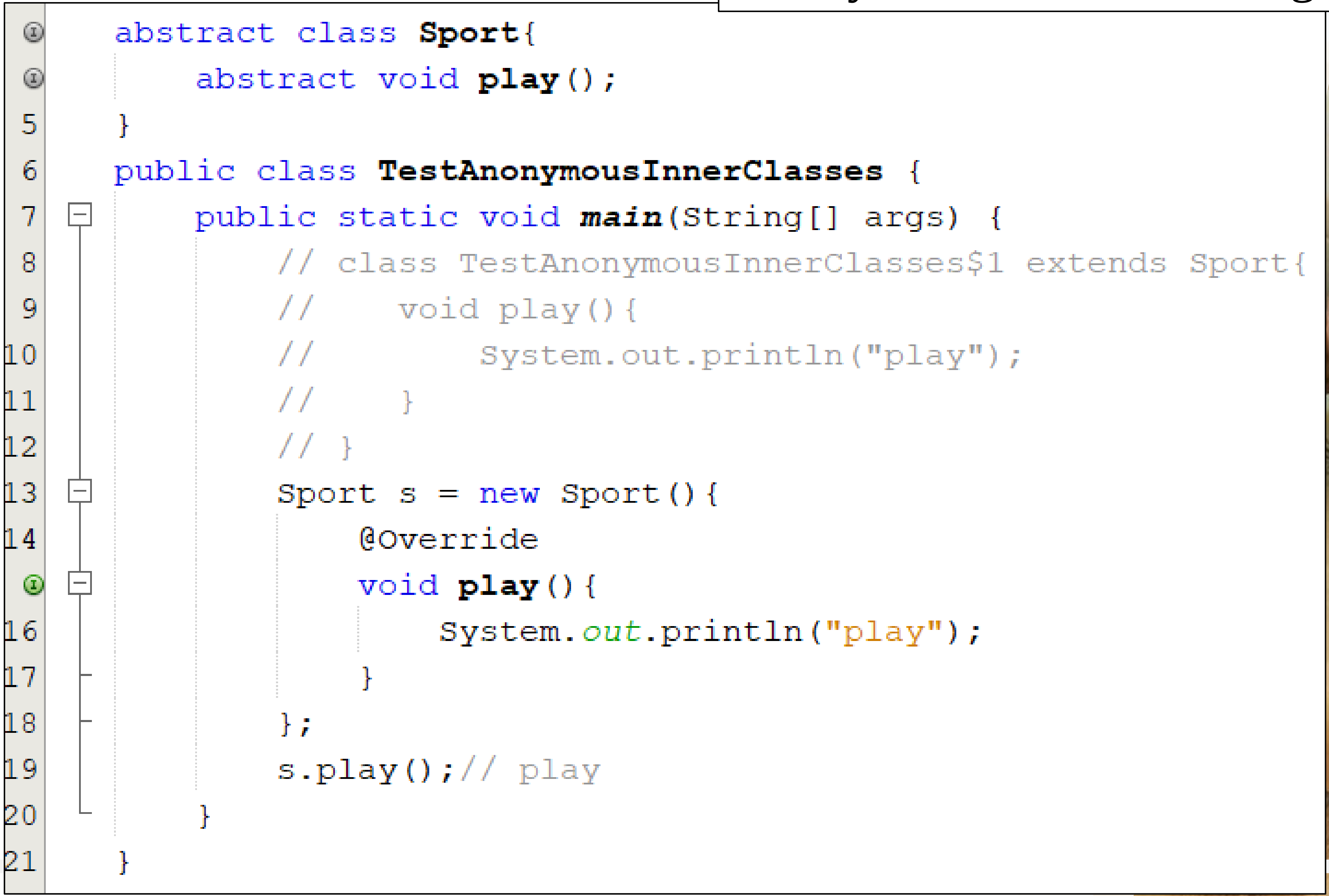


Anonymous class implementing an interface



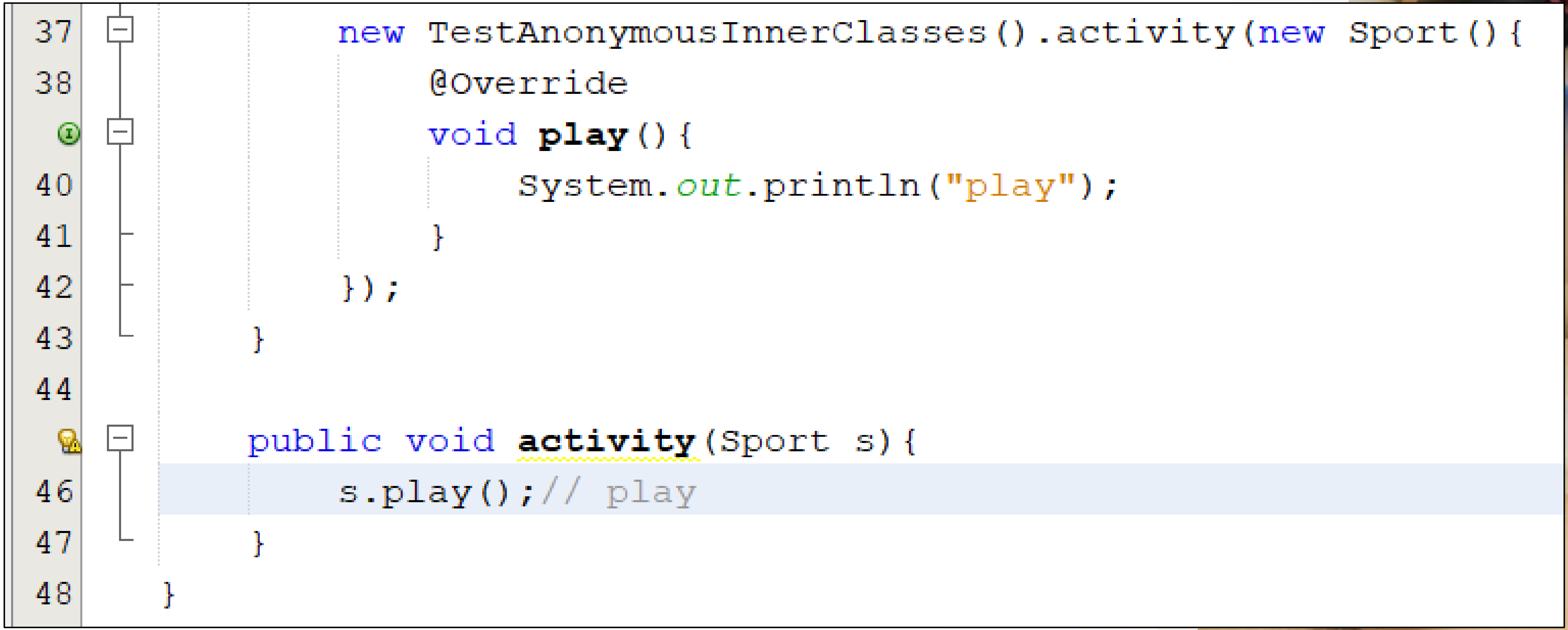
```java
interface Restable{
    void rest();
}
public class TestAnonymousInnerClasses {
    public static void main(String[] args) {
        // class TestAnonymousInnerClasses$1 implements Restable{
        //     public void rest(){
        //         System.out.println("rest");
        //     }
        // }
        Restable r = new Restable(){
            @Override
            public void rest(){
                System.out.println("rest");
            }
            public void sleep(){}
        };
        r.rest();// rest
        r.sleep();// no sleep() method in Restable
    }
}
```

Anonymous class extending a class



```java
abstract class Sport{
    abstract void play();
}
public class TestAnonymousInnerClasses {
    public static void main(String[] args) {
        // class TestAnonymousInnerClasses$1 extends Sport{
        //     void play(){
        //         System.out.println("play");
        //     }
        // }
        Sport s = new Sport(){
            @Override
            void play(){
                System.out.println("play");
            }
        };
        s.play();// play
    }
}
```

Anonymous class passed as a method argument



```java
        new TestAnonymousInnerClasses().activity(new Sport(){
            @Override
            void play(){
                System.out.println("play");
            }
        });
    }

    public void activity(Sport s){
        s.play();// play
    }
}
```