Java 17

the cool new features since Java 11

Java 17

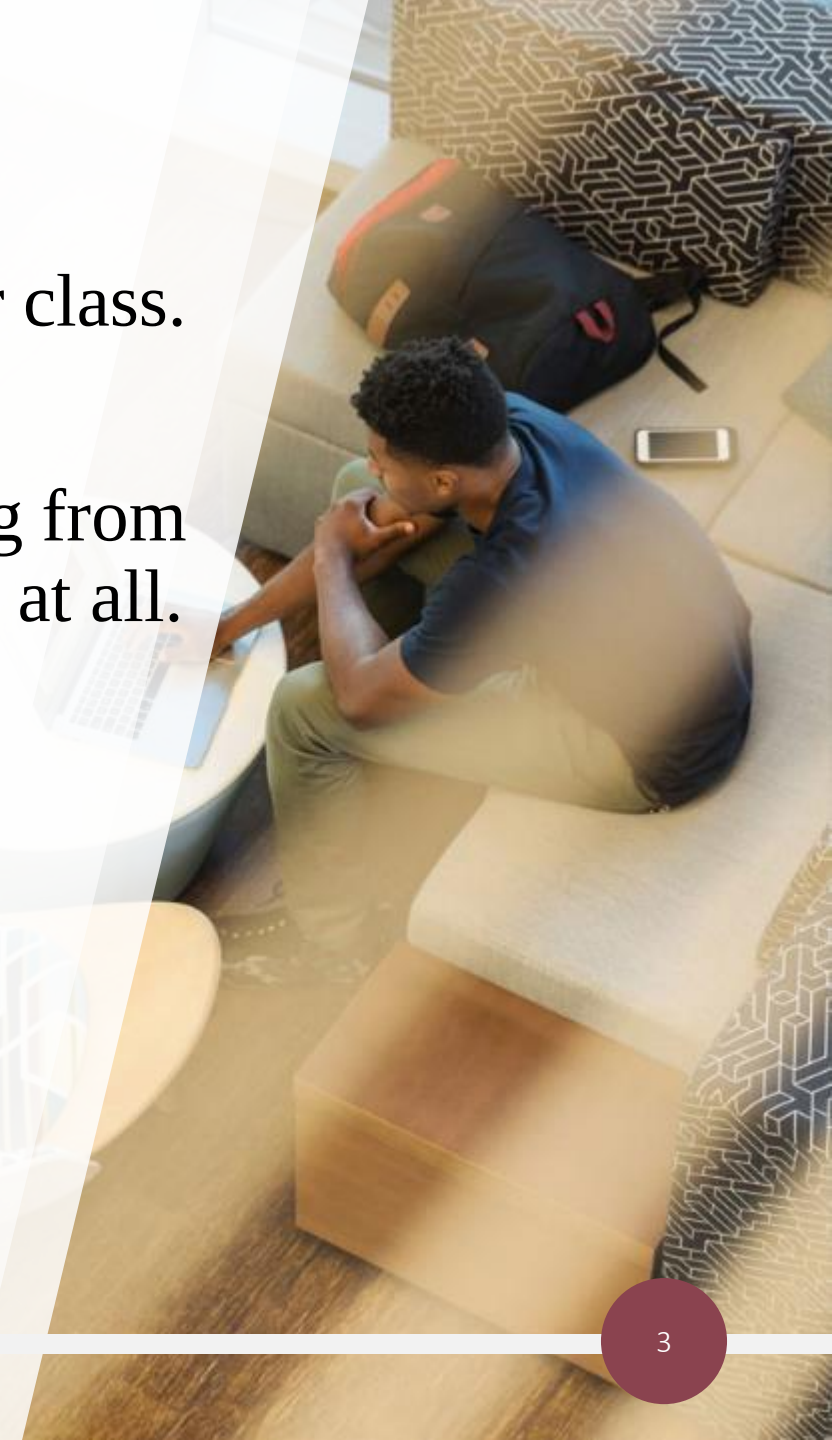
- Java 17 is a Long Term Support (LTS) release.
  - https://openjdk.org/projects/jdk/17/

- Topics:
  - sealed classes.
  - records.
  - *switch* expressions and pattern matching.
  - text blocks

- Assignment

Sealed classes

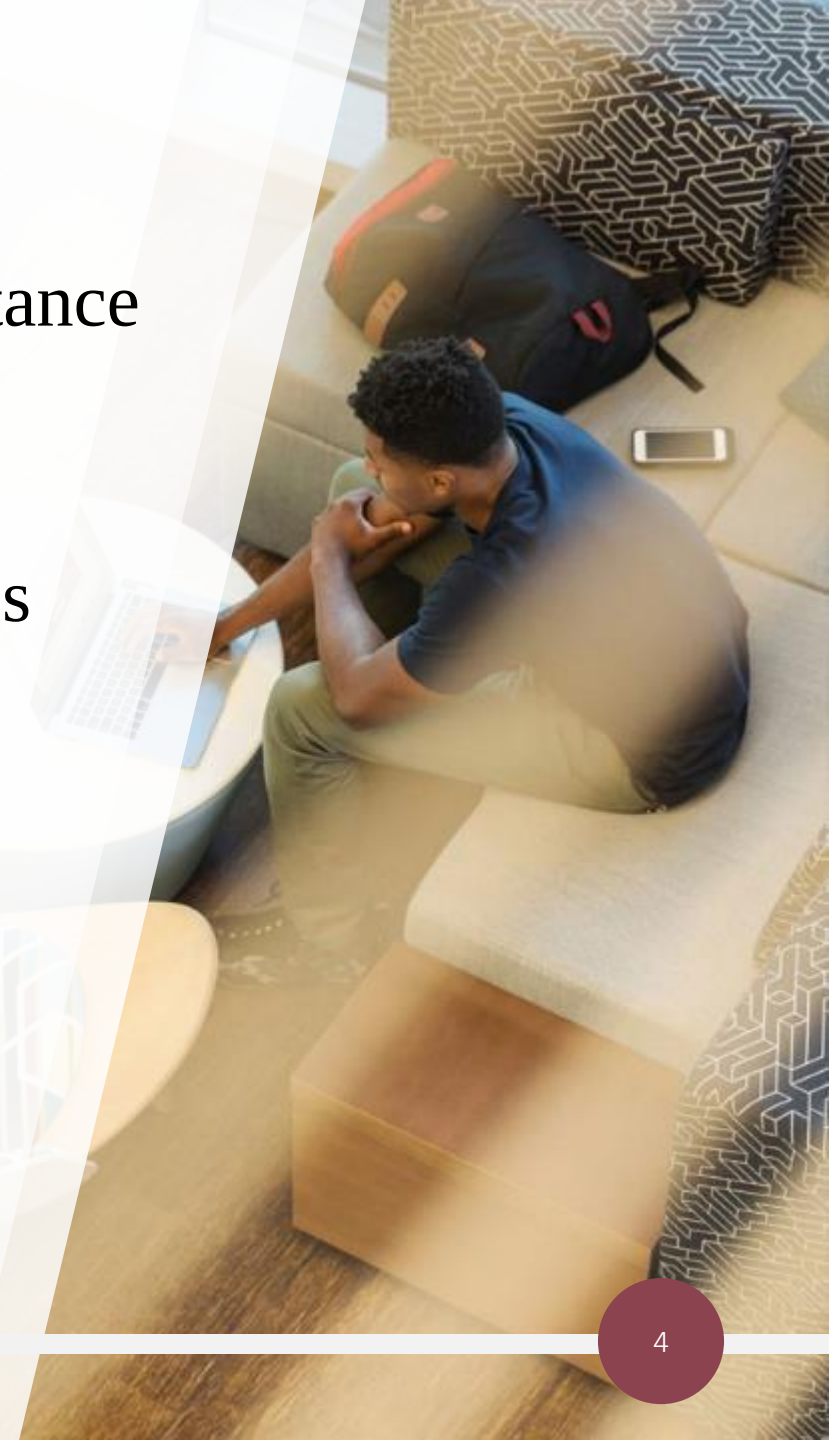
- Inheritance enables any class to inherit from any other class.
- Making a class *final* prevents any class from inheriting from that class i.e. the *final* class cannot become a super type at all.
- What if you wanted your class to be available for inheritance but only to certain classes?



Sealed classes

- Sealed classes enable us to control the scope of inheritance by enabling us to specify a classes' subtypes.

- Also works with interfaces (we can define what classes *implement* the interface).



Records

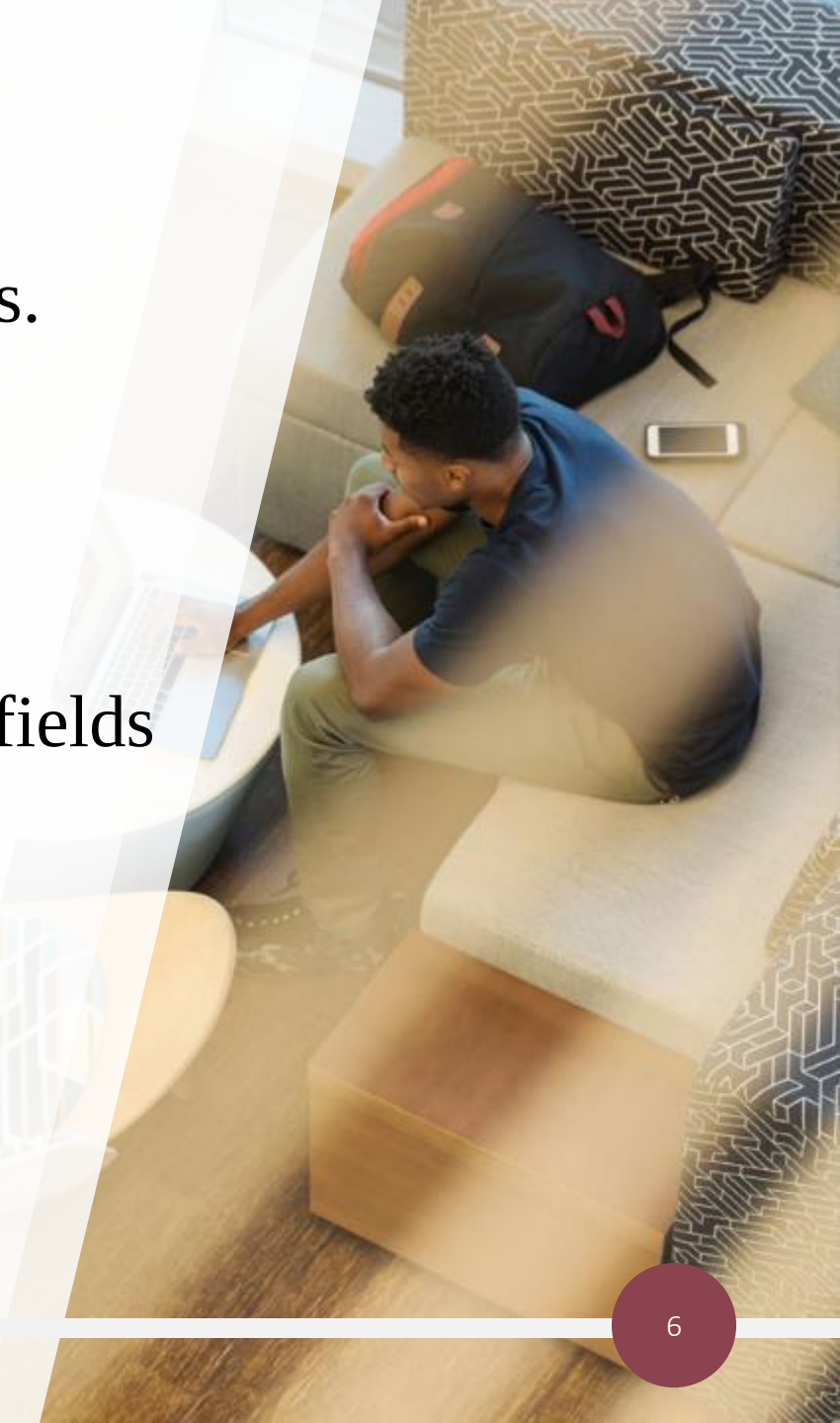
- Records are a special type of class that help avoid boilerplate code. They are considered “data carriers”.

- Records are immutable and are *final* by default.

- You cannot extend your custom record because records already (implicitly) extend from the *Record* class. This is similar to enums (which implicitly extend from *Enum*).

Records

- Records can have both *static* fields and *static* methods.
- Records can have instance methods.
- Records cannot have instance fields. All the instance fields are listed as “components” in the record declaration.
- Records can implement interfaces.

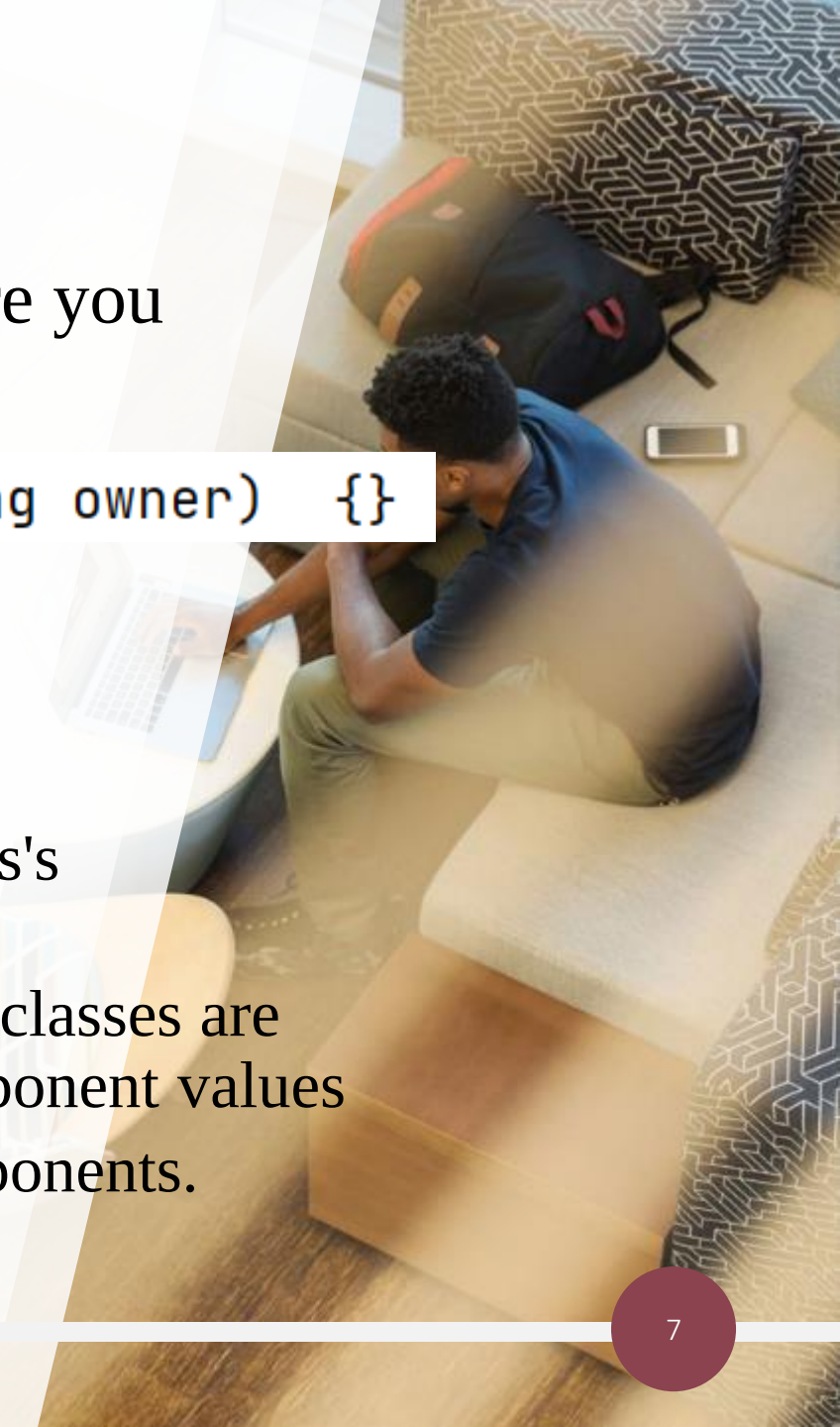


Records

- Records are specified using a record declaration where you specify the “components” of the record.

```
public record CarRecord(String regNumber, String owner)  {}
```

- Implicitly generated are:
  - canonical constructor
  - *toString()* - the string representation of all the record class's components, with their names.
  - *equals()* and *hashCode()* - which specify that two record classes are equal if they are of the same type and contain equal component values
  - *public* accessor methods with the same name as the components.



Records

- You can override all the default implementations. This includes the canonical constructor (perhaps for data validation).

```
// default canonical constructor
3 usages
public CarRecord(String regNumber, String owner)  {
    this.regNumber=regNumber;
    this.owner    =owner;
}
```

```
// custom canonical constructor
3 usages
public CarRecord(String regNumber, String owner)  {
    if(regNumber.length() <= 4){
        throw new IllegalArgumentException();
    }
    this.regNumber=regNumber;
    this.owner    =owner;
}
```

- Compact constructor is a concise variation of the canonical constructor and is specific to records.

```
public record CarRecord(String regNumber, String owner)  {}
```

```
// custom canonical constructor
3 usages
public CarRecord(String regNumber, String owner)  {
    if(regNumber.length() <= 4){
        throw new IllegalArgumentException();
    }
    this.regNumber=regNumber;
    this.owner    =owner;
}
```

```
// compact constructor - specific to records
3 usages
public CarRecord  {
    if(regNumber.length() <= 4){
        throw new IllegalArgumentException();
    }
}
```

Pattern-matching *switch* statements

- The *switch* statement has undergone several changes over the years.

- *switch* expressions were introduced as a “preview feature” in Java 12 and became permanent in Java 14.
  - a “preview feature” is a feature that is designed and implemented but not yet permanent (and may never be); it allows a large developer audience to test out the feature before committing to it.

- *yield* statement introduced in Java 13 to support *switch* expressions.

Pattern-matching *switch* statements

- This is a “preview feature” in Java 17.

- Preview features are, by default disabled so you must explicitly enable them.

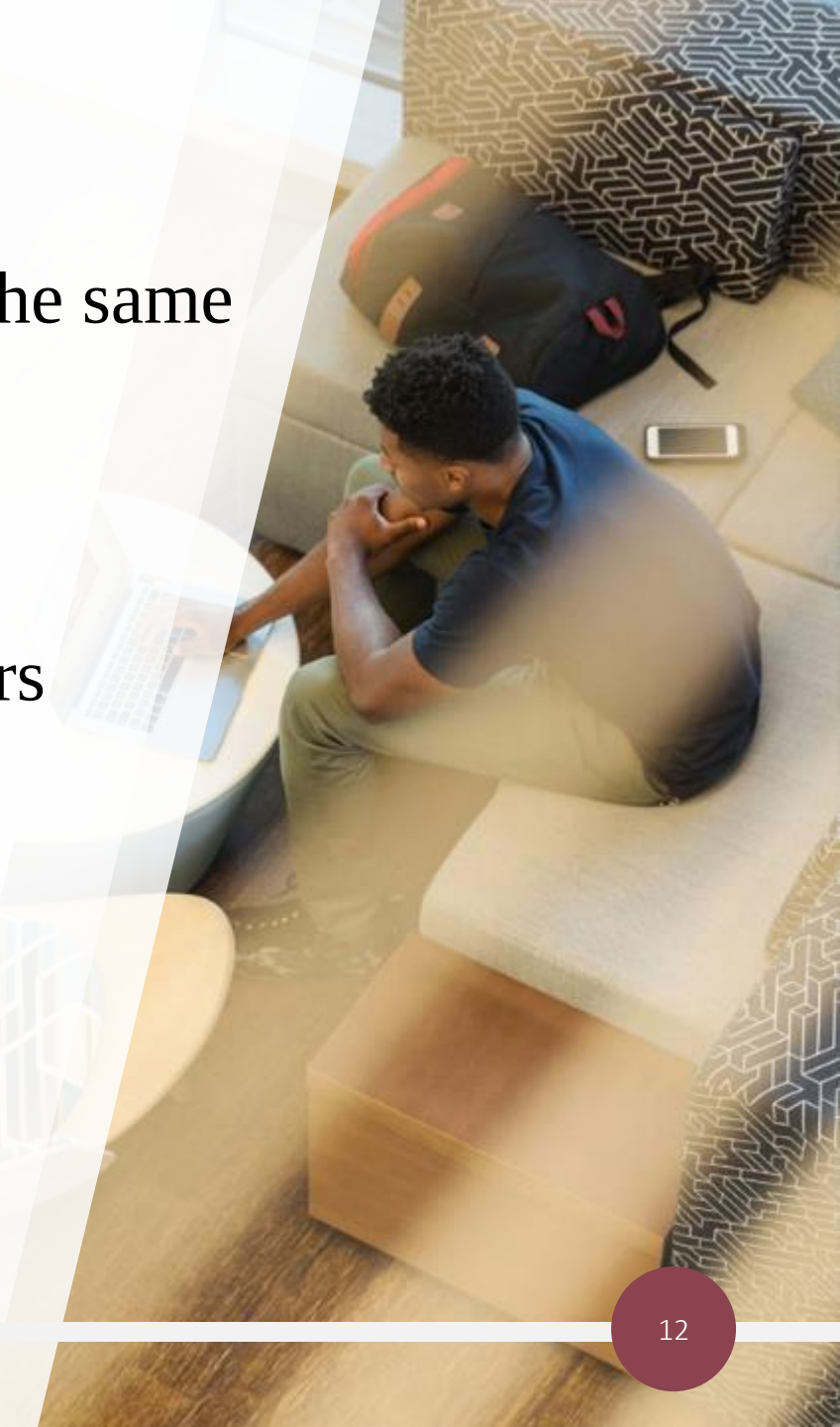
Text Blocks

- A text block is a *String* object and as a result, shares the same properties as *String* objects (immutable and interned).
  - you can call *String* methods on a text block.

- A text block begins with three double-quote characters followed by newline i.e. `"""`
  - text blocks cannot be put on one line
  - the text of a text block cannot follow the `"""`

- Example:

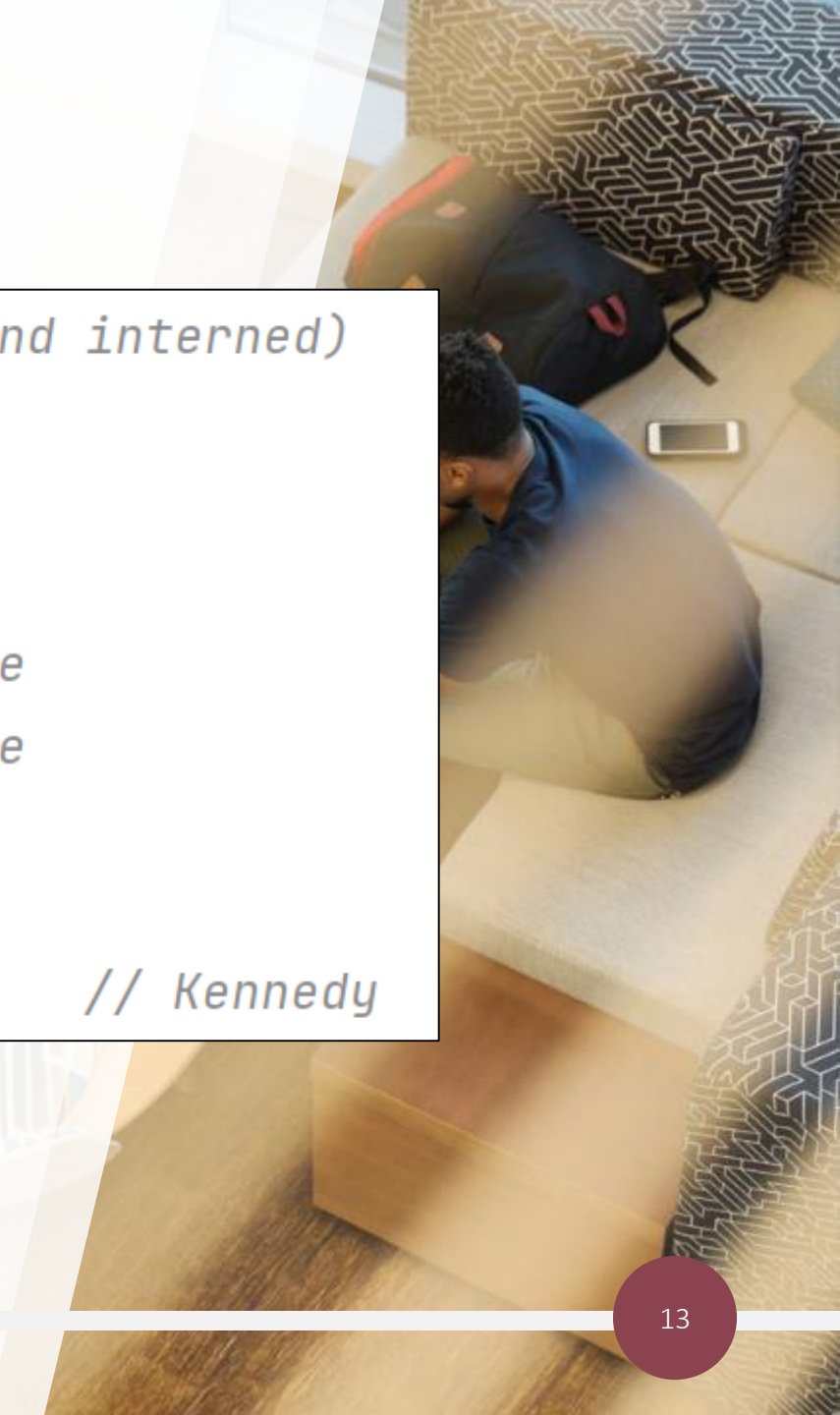
```
String tbName = """
        Sean Kennedy""";
```



Text Blocks

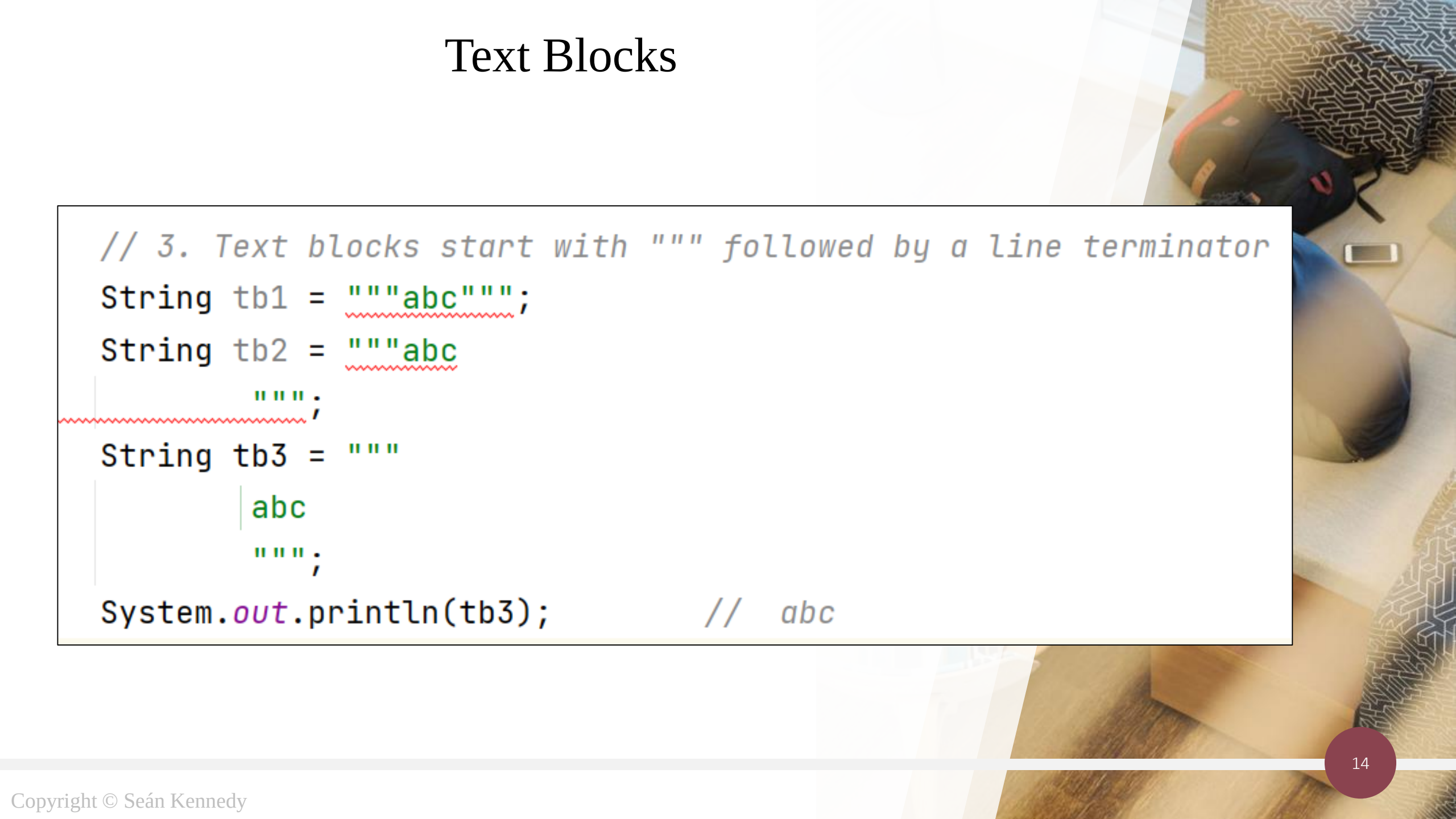
```
// 1. A text block is a String object (immutable and interned)
String sName  = "Sean Kennedy";
String tbName = """
        Sean Kennedy""";
System.out.println(sName.equals(tbName));   // true
System.out.println(sName == tbName);        // true

// 2. String methods can be applied to text blocks
System.out.println(tbName.substring( beginIndex: 5));    // Kennedy
```



Text Blocks

```
// 3. Text blocks start with """ followed by a line terminator
String tb1 = """abc""";
String tb2 = """abc
             """;
String tb3 = """
             abc
             """;
System.out.println(tb3);        //  abc
```



Text Blocks

- In *String* literals, embedded quotes must be escaped. This is not the case with text blocks.

```
// 4. Embedded double quotes are not required in text blocks
String sQuote = "Hamlet: \"There is nothing either good or bad, " +
        "but thinking makes it so\""; // on one line
System.out.println(sQuote);
String tbQuote = """
        Hamlet: "There is nothing either good or bad, but thinking makes it so"
        """;
System.out.println(tbQuote); // on one line
```

```
Hamlet: "There is nothing either good or bad, but thinking makes it so"
Hamlet: "There is nothing either good or bad, but thinking makes it so"
```

Text Blocks

- Depending on where you place the closing delimiter (the three double quotes), determines whether or not you have a closing “\n”.

```
String tbBookTitle1 = """
        Java
        Memory
        Management
        """;    // same as "Java\nMemory\nManagement\n"; // newline at end
```

```
String tbBookTitle2 = """
        Java
        Memory
        Management""";  // same as "Java\nMemory\nManagement"; // NO newline at end
```

Text Blocks

- Spacing is determined by the closing delimiter position or the first non-space character, whichever is encountered first.

- All spaces (known as incidental spaces) leading up to that position, are stripped from all lines in the text block.

- Note that, the above algorithm only works, if the closing delimiter is on a line of it’s own.