# Records

## Overview

This lab will give you an opportunity to define and use a record type in Java. You'll also learn about how to use the Builder Pattern to build objects that have lots of (possibly optional) fields.

## Step 1: Getting started

In IntelliJ, create a new module named student.records. In the src folder, create a new package with a name of your choice. In the package, define a new Java record named Book to hold the following fields (i.e., "components"):

* String title
* String genre
* String author
* java.time.Year published
* String description

Now define a "main" Java class and write some code to create a Book object and display it on the console. Java records have a default toString() method that comes in handy here (you can customize toString() if you like).

## Step 2: Using the Builder Pattern

If you have a Java record with a lot of fields, you have to pass in values for all these fields when you create an object. This can get overwhelming if the Java record has a crazy number of fields!

You can use the Builder Pattern to help you in this case. In the Builder Pattern, you define a helper class (a "builder" class) that allows you to feed in just the values you're interested in, and leave the other ones blank. When you've specified all the fields you're interested in, you then tell the builder object to "build" the object you're really interested in (e.g., a Book object).

Follow these steps to define a "builder" class for books:

* Inside the Book record, define a public static nested class named Builder (for example).
* In the Builder class, define private instance variables for all the fields that will eventually be fed into a Book object.
* For each of these fields, define a public method that the client can call to specify a value for the field. For example:
  + Define a public method named title().
  + The method should receive a String parameter, specifying the book's title.
  + Inside the method, copy this parameter into the Builder's title field.
  + At the end of the method, return the Builder object itself (i.e., return this). This enables the client code to call Builder methods in a cascaded fashion, as we'll show shortly.
* Repeat the above step for all the fields in the Builder class. You'll end up with 5 methods that look something like this:
  + public Builder title(String title) {…}
  + public Builder genre(String genre) {…}
  + public Builder author(String author) {…}
  + public Builder published(Year published) {…}
  + public Builder description(String description) {…}
* The next step is to define a method that the client can invoke when they've finished feeding in book info and they’re ready to actually create a Book object. This method is typically named build(). Implement the method as follows:

public Book build() {

return new Book(title, genre, author, published, description);

}

* You can now use the Builder class as follows in your main code, to create a Book object containing just the field values you want to specify. For example:

// Create a Builder object and feed in fields we're interested in.

Book.Builder builder = new Book.Builder()

.title("When the Lion Feeds")

.genre("Adventure")

.author("Wilbur Smith") // I'd recommend this author!

.published(Year.of(1980));

// Create a book object, by using the builder.

Book book1 = builder.build();

System.out.println(book1);

## Step 3 (if time permits): Validate the book fields

Builder classes typically perform validation when the client code calls build(), to ensure that important fields have been specified. Enhance your Builder class's build() method so that it performs the following validation tests (for example):

* title - must not be null, max length is 100 characters
* genre - can be null, but if specified then the max length is 20 characters
* author - can be null, but if specified then the max length is 50 characters
* published - must not be after the current year
* description - can be null, but if specified then the max length is 500 characters

If any of these validation tests fail, throw an exception and do not create a Book object.