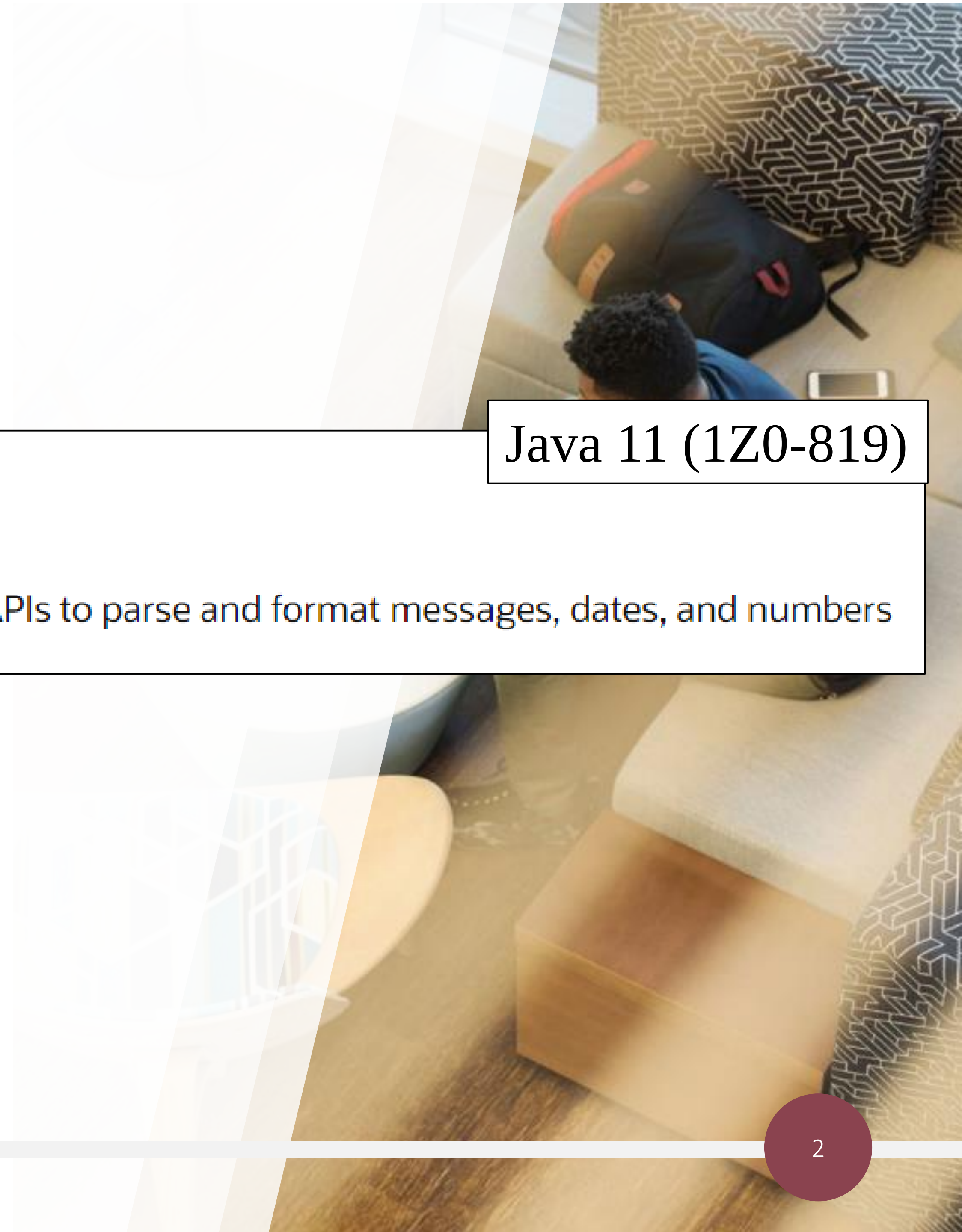
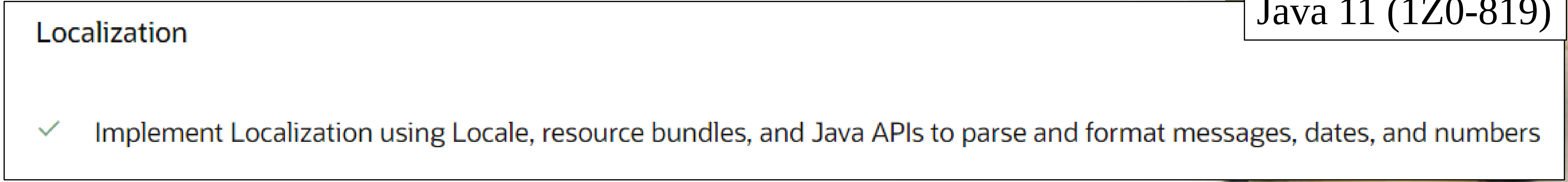
Localisation

Localisation



Java 11 (1Z0-819)



Localization

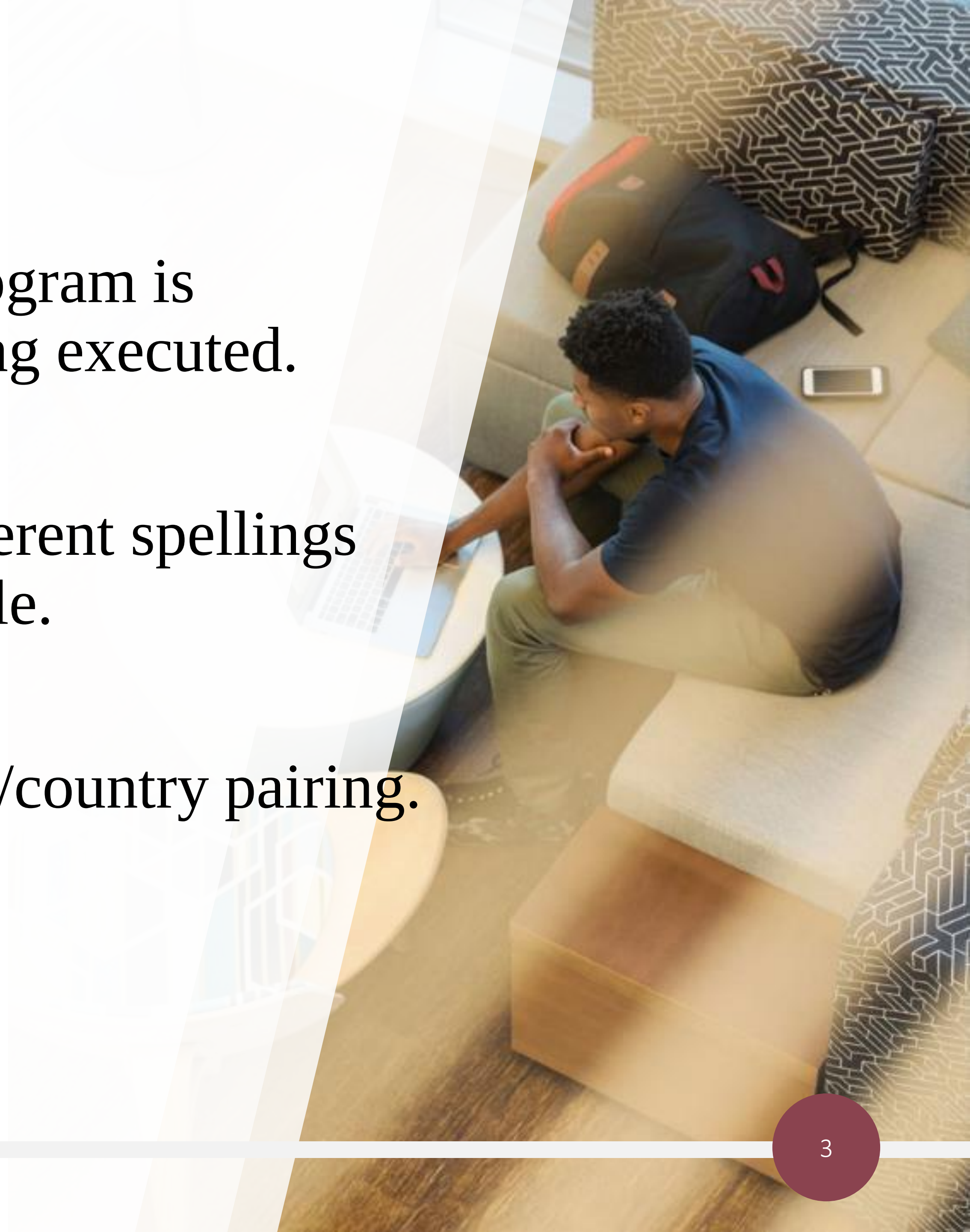
✓ Implement Localization using Locale, resource bundles, and Java APIs to parse and format messages, dates, and numbers

Localisation

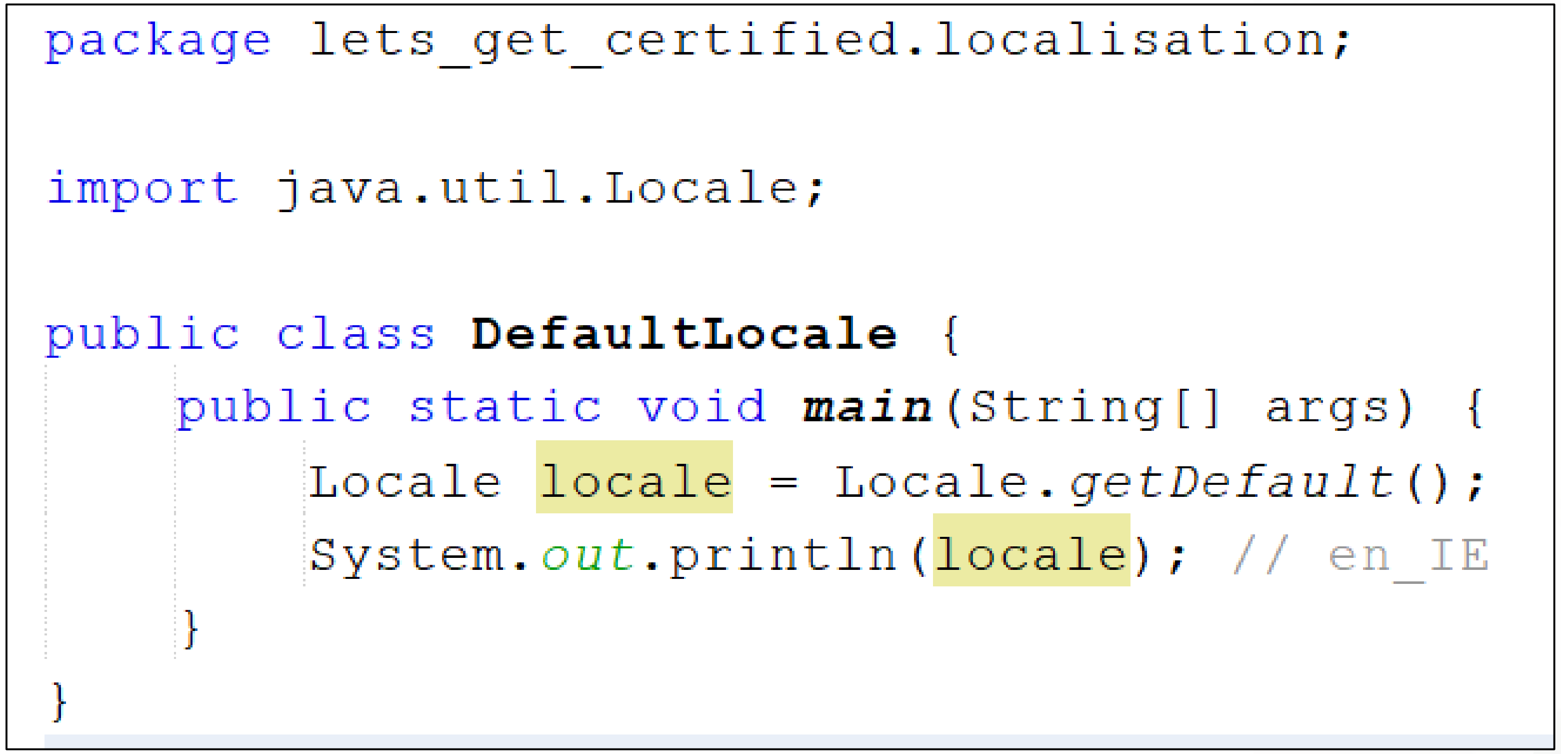
- Localisation involves ensuring that your program is adaptable to the location from which it is being executed.

- This involves translating strings, use of different spellings and formatting of dates/numbers for that locale.

- A locale, in its simplest terms, is a language/country pairing.



Localisation



```java
package lets_get_certified.localisation;

import java.util.Locale;

public class DefaultLocale {
    public static void main(String[] args) {
        Locale locale = Locale.getDefault();
        System.out.println(locale); // en_IE
    }
}
```

- Locale format: language_COUNTRY
  - language is in lowercase and is mandatory
  - COUNTRY is in capitals and is optional
  - “l_C” = “**l**ake **C**omo”, “**l**iz **C**heney”, “**l**eaving **C**ert”

Localisation

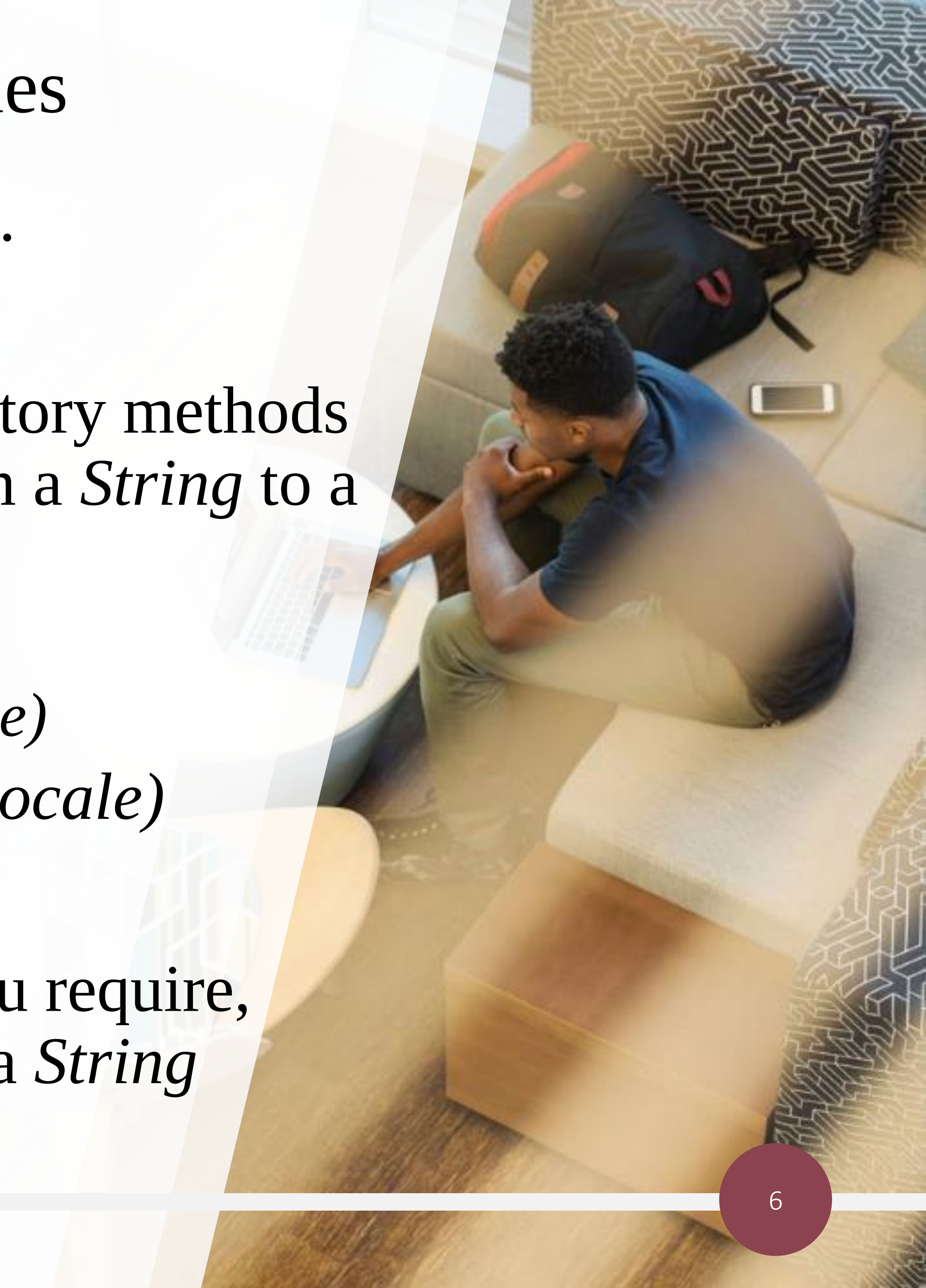
- To create/select a locale that is not the default locale, there are 3 popular options:
  - constructors
    - pass in a language only or both a language and a country
  - built-in constants
    - the *Locale* class provides constants for the most popular locales
  - builder design pattern
    - flexible - specify the properties you want, in any order
    - locale is built at the end

Localising Numbers and Currencies

- Currencies and numbers differ between locales.

- The *NumberFormat* class has several *static* factory methods than enable us to both in both directions i.e. from a *String* to a number and a number to a *String*.
  - *getInstance(), getInstance(locale)*
  - *getNumberInstance(), getNumberInstance(locale)*
  - *getCurrencyInstance(), getCurrencyInstance(locale)*

- Once you have the *NumberFormat* instance you require, you can invoke *format()* to convert a number to a *String* and *parse()* to convert a *String* into a number.

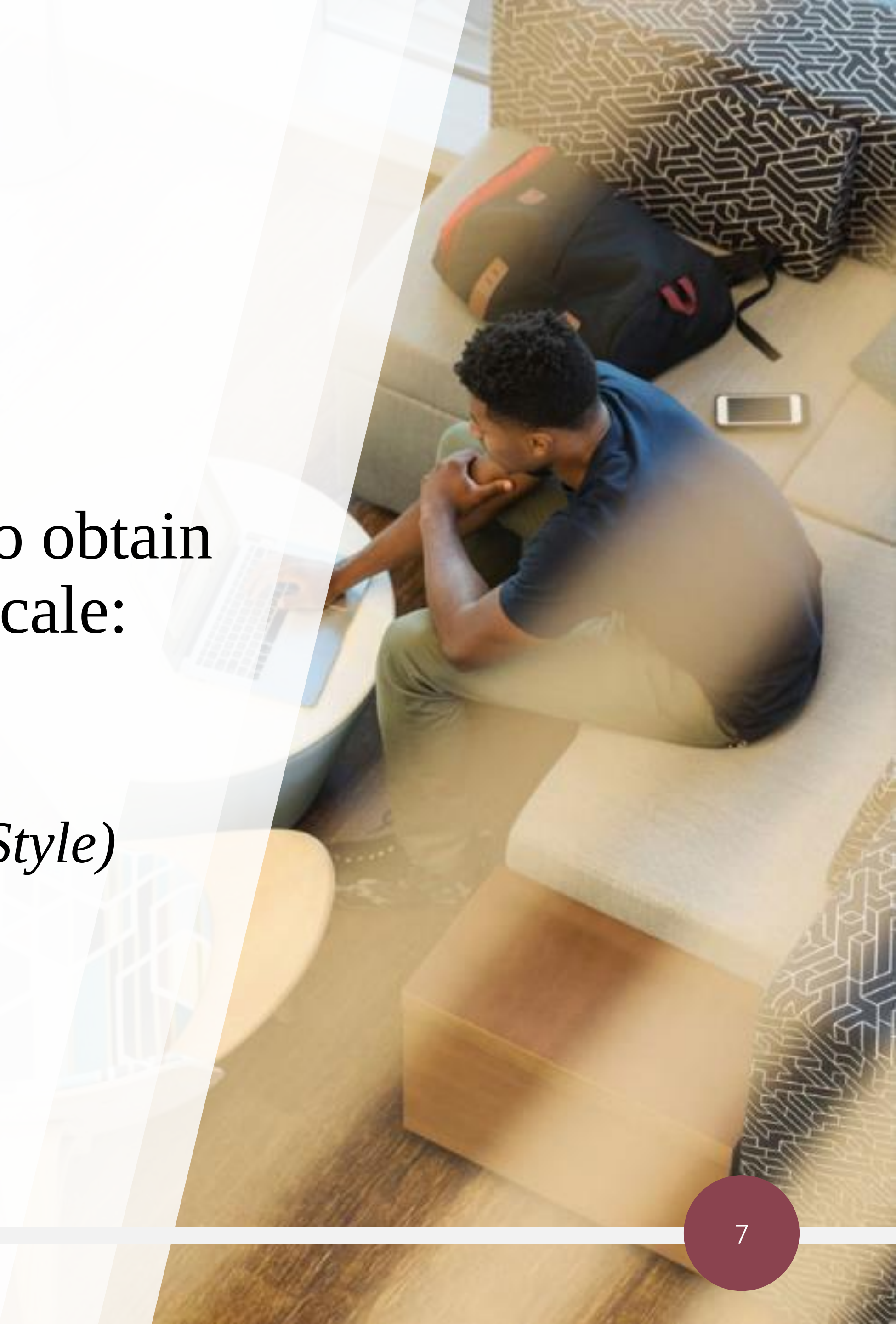


Localising Dates

- Date formats vary by locale.

- *DateTimeFormatter* contains factory methods to obtain formatters for dates (and times) for the current locale:
  - *DateTimeFormatter.ofLocalizedDate(dateStyle)*
  - *DateTimeFormatter.ofLocalizedTime(timeStyle)*
  - *DateTimeFormatter.ofLocalizedDateTime(dateTimeStyle)*

- To customise for a specific locale, append "*withLocale(locale)*".



Category enums

- When you change the default locale with *Locale.setDefault()*, certain options regarding display and formatting are set automatically.

- We can set these options individually ourselves using the *Locale.Category* enums: DISPLAY and FORMAT.
  - DISPLAY – relates to display information.
  - FORMAT – formatting currencies, dates and numbers.

Resource Bundles

- Resource bundles contain locale-specific data.

- They take the format of a map with keys and values. The values are locale-specific.

- Resource bundles can be created in 2 ways:
  - java files – extend *ListResourceBundle*.
  - property files – text files with key value pairs.

- The exam focuses on properties.

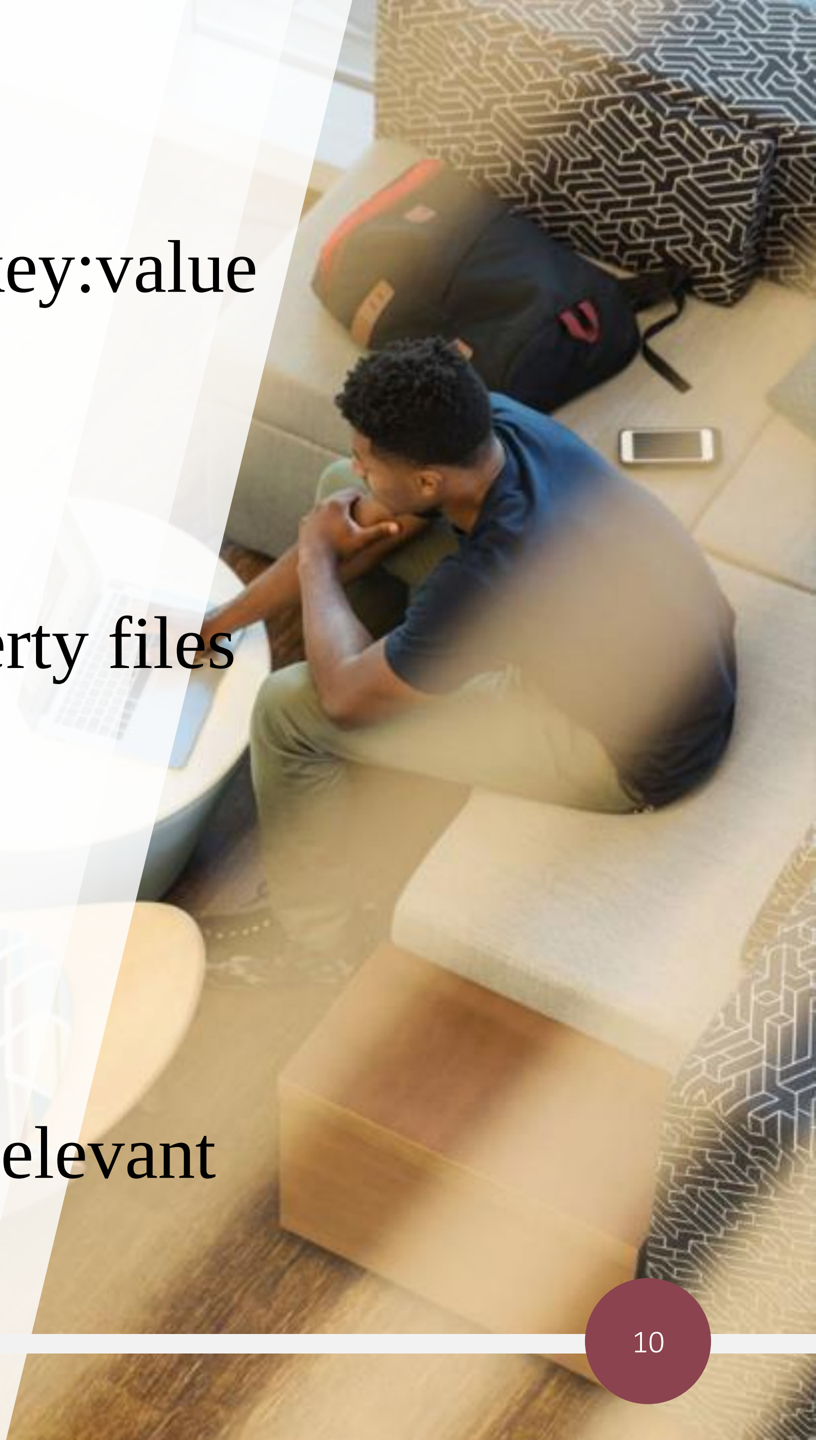
Property Files

- Simple text files in the format of key=value pairs (or key:value pairs). The keys and values are both Strings i.e. Map<String,String>.

- Several locales imply several property files. The property files therefore, follow very specific naming conventions.

- The suffix is *.properties*.

- *ResourceBundle.getBundle(name, [locale])* loads the relevant property file.



Resource Bundle Search Order

- Resource Bundle Name: *Team*.
- Required locale: *de_DE* (German in Germany).
- Default locale: *en_IE* (English in Ireland).

• Team_de_DE.properties	// exactly what we are looking for
• Team_de.properties	// take off the country
• Team_en_IE.properties	// try the default locale
• Team_en.properties	// take off the country
• Team.properties	// try the default bundle
• MissingResourceException	// ☹

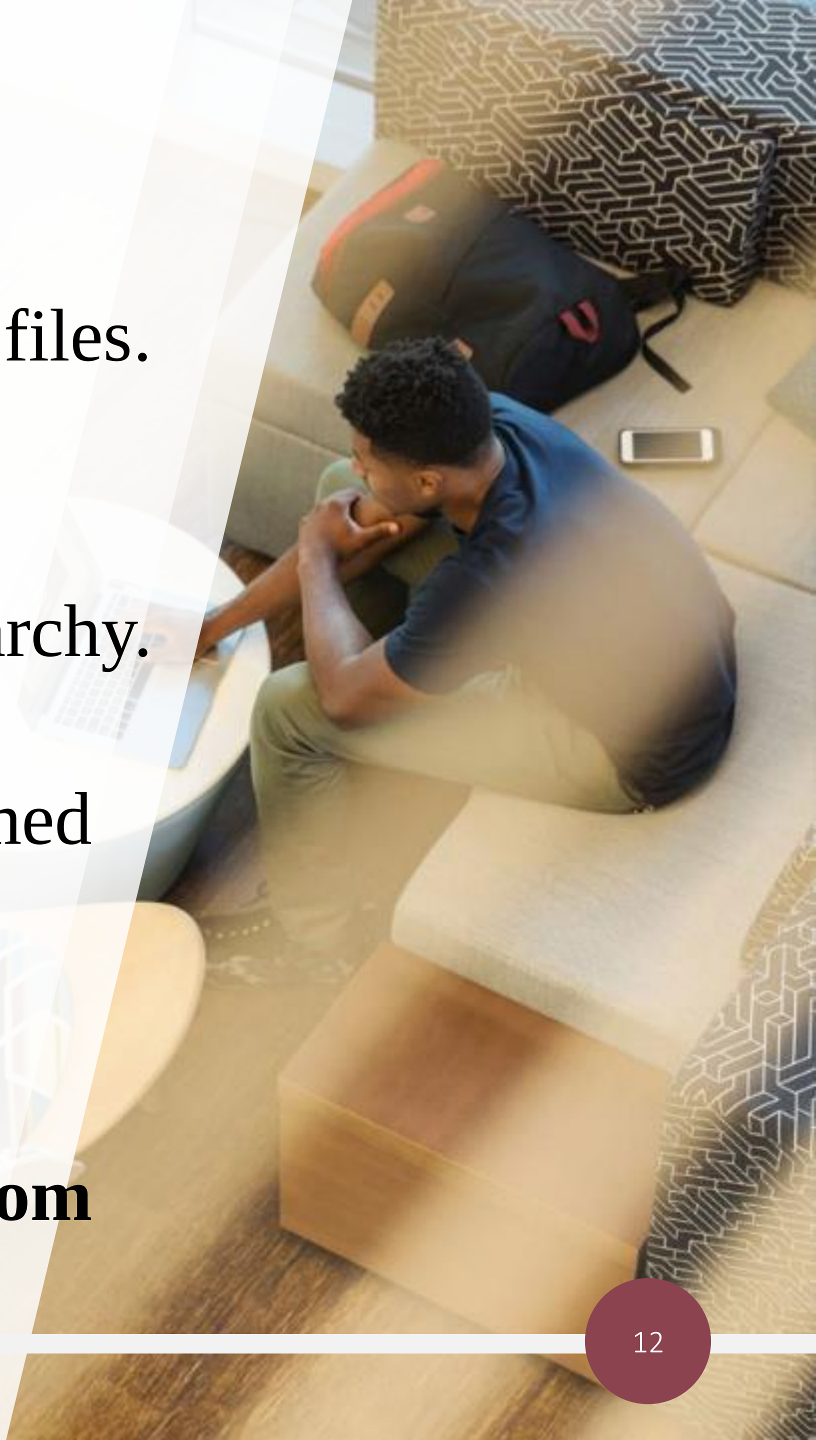
Resource Bundle Search Order

- All of the keys do not have to be specified in all of the properties files. This avoids duplication across property files.

- The properties files can inherit from each other; more specifically, you can inherit from any *parent* in the hierarchy.

- A parent resource bundle name in the hierarchy is formed by removing components of the name until the top is reached. The top is the default bundle.

- **Once a resource bundle is found, only properties from that hierarchy will be used.**



Resource Bundle Search Order

Resource Bundle Name: Team	Required locale: de_DE	Default locale: en_IE
First Search order:		Team_de_DE.properties Team_de.properties Team_en_IE.properties Team_en.properties Team.properties
Found: Team_de_DE.properties		Search order set to: Team_de_DE.properties Team_de.properties Team.properties (default locale irrelevant)
Found: Team_en_IE.properties		Search order set to: Team_en_IE.properties Team_en.properties Team.properties (de_DE locale irrelevant)