COP3330C Module 8 Practice Exercise

In this practice exercise we will implement the decorator pattern. We will use this pattern to add a feature to our HappyBirthday application which localizes the greeting based on the recipients locale.

Design Notes

We want to localize the date and greeting for a user's birthday card as a new feature for the BirthdayCard application without impacting our "legacy" code; in other words, we need to allow prior users to continue using the application with a non-localized greeting but allow new users to include their locale as part of their profile and provide them a greeting using the appropriate formatting and translation based on the locale. The decorator pattern allows us to do this by wrapping the existing User class with a locale-based User decorator, and by wrapping the existing BirthdayCard class with a localized version.

Note that the code to implement the abstract factory pattern with different age categories has been removed for this exercise, we are working with a stripped-down version of the original code from the last module.

A new class which derives from User has been added; this UserWithLocale class includes a User reference and a Locale field:

```
public class User {
    private StringBuilder name;
    private String email;
    private ZonedDateTime birthday;
…
class UserWithLocale extends User {
    private User user;
    private Locale locale;
```

An overloaded constructor for the new class allows instantiation using a previously created User and their desired locale:

```
public UserWithLocale(User user, Locale locale) {
    this.user = user;
    this.locale = locale;
}
```

Similarly, a class is derived from the BirthdayCard class which represents a localized Birthday Card:

```
class BirthdayCard_Localized extends BirthdayCard {
    private BirthdayCard birthdayCard;
```

In the overloaded constructor for this subclass, we instantiate using a non-localized card, then reset the message to a default value which will be used to create the localized greeting:

```
public BirthdayCard_Localized(BirthdayCard birthdayCard) {
    this.birthdayCard = birthdayCard;
    // non-localized card will have bordered message, reset it here
```

```
        birthdayCard.message = "Happy Birthday, " +
                birthdayCard.user.getName() + "!";
        this.buildCard(birthdayCard.getUser(), birthdayCard.getMessage());
    }
```

In the main method, we use the following logic to test our changes:

Create an array of “legacy” users (no locale specified)
Create and send non-localized cards to these users.

Create an array of new locale-based users (add a locale to the previous users)
Create and send localized cards to these users.

To create the new localized users we use pass the original users and a locale to the derived locale-based User class’s overloaded constructor:

```
final UserWithLocale[] USERSWITHLOCALES = {
        new UserWithLocale(USERS[0], new Locale("en")),
        new UserWithLocale(USERS[1], new Locale("en")),
        new UserWithLocale(USERS[2], new Locale("fr")),
        new UserWithLocale(USERS[3], new Locale("de")),
        new UserWithLocale(USERS[4], new Locale("zh"))
};
```

Polymorphism allows us to use much of the existing code without changes, for instance when we add the users to the birthdays ArrayList, we can use the addBirthdays method for both legacy and new users:

```
public void addBirthdays(User... users) {
    for (User u : users) {
        birthdays.add(u);
    }
}
```

We make a distinction between old and new cards when we instantiate them:

```
BirthdayCard card;
// decorate and send a legacy card
if (u instanceof UserWithLocale)
    card = new BirthdayCard_Localized(new BirthdayCard(u));
else
    card = new BirthdayCard(u);
```

Note that we instantiate an anonymous non-localized card to pass to the localized card constructor.

For the localization, we use simple Happy Birthday greeting with the current date.

We use a DateTimeFormatter which is customized by locale and applied to the user's ZonedDateTime property. There are a variety of styles we could apply to the date, I've chosen the basic FormatStyle

enum value of MEDIUM which includes the name of the month (https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/time/format/FormatStyle.html)

For the localization of the birthday greeting, there are four property files provided in the solution in the form of a "Birthday" resource bundle:

Birthday_en.properties -- US English birthday greeting
Birthday_fr.properties -- French birthday greeting
Birthday_de.properties -- German (Deutsch) birthday greeting
Birthday_zh.properties --Chinese (Mandarin) birthday greeting

Examine the contents of these files to see how the greeting property is implemented and note the location of the files. If you create the files in a package, IntelliJ will recognize the file extensions and group them as a resource bundle for you.

Creating a property is easy since they are simple key/value combinations, but some research is required to determine the value (see below). This exercise is a very trivial implementation, we are translating only the "Happy Birthday" greeting. A full birthday card greeting would require much more expertise in the grammatical forms of each language; developers must tread carefully here to avoid awkward or inappropriate translations. We usually depend on a localization group (either in-house or contracted) to do this for us since language fluency is required. Even if you have had a few courses in a foreign language or speak a "smattering" of a second language, do not volunteer for this task; trust me on this.

The German and French translations of "Happy Birthday" are simple to find on the Web:

German: Alles Gute zum Geburtstag
French: Joyeux Anniversaire

The Chinese version takes a little more work since we have to use Unicode values to represent the phrase in Mandarin (or "Hànzì" as the symbols are called). I used the following sites:

https://chinainternshipplacements.com/blog/happy-birthday-in-chinese-mandarin/
https://www.chineseconverter.com/en/convert/unicode

which gave me the following:

生日快乐
\u751f \u65e5 \u5feb \u4e50

Here's the code, in the buildCard method in the localized BirthdayCard class:

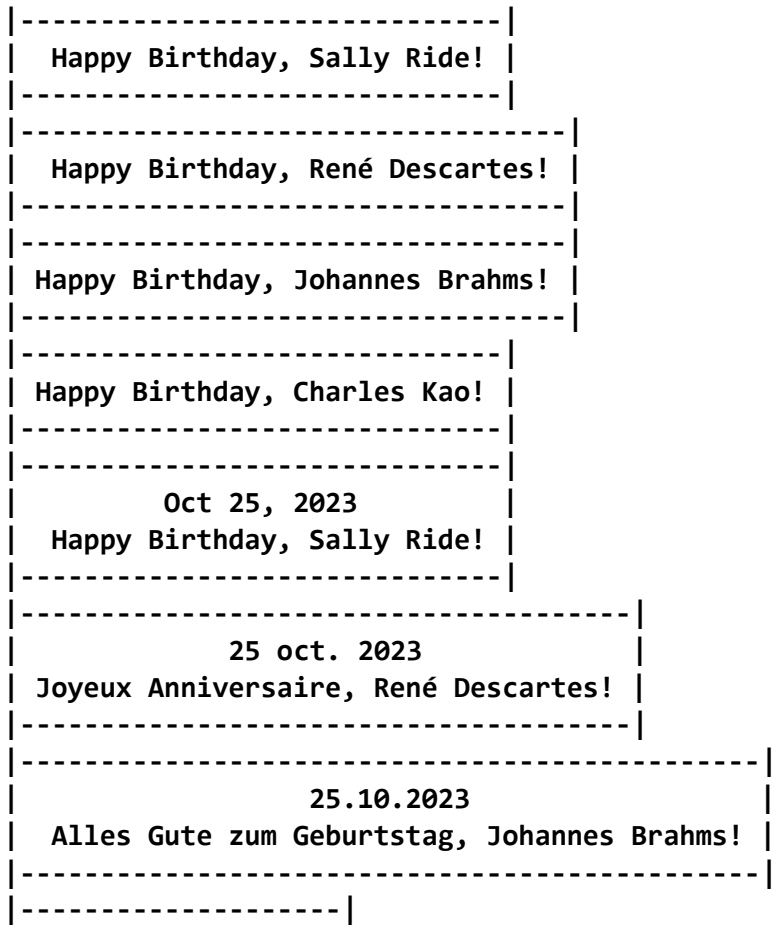
```
// load the property and create the localized greeting
ResourceBundle res = ResourceBundle.getBundle(
        "edu.fscj.cop3330c.birthday.Birthday",
        ((UserWithLocale)getUser()).getLocale());
```

```
String happyBirthday = res.getString("HappyBirthday");
message = message.replace("Happy Birthday", happyBirthday);

// format and display the date
DateTimeFormatter formatter =
        DateTimeFormatter.ofLocalizedDate(FormatStyle.MEDIUM);
formatter =
        formatter.localizedBy(((UserWithLocale)getUser()).getLocale());
String dateStr = getUser().getBirthday().format(formatter) + "\n";

// add the localized greeting
msg = dateStr + message;
```

Notice that if we don't find the resource bundle, we have a safety net in the catch clause which simply uses the English version of the greeting.

```
} catch (java.util.MissingResourceException e) {
    System.err.println(e);
    msg = "Happy Birthday, " + getUser().getName() + "!";
}
```

Here is the output from the program (4 non-localized followed by 4 localized):



```
|     2023年10月25日     |
| 生日快乐, Charles Kao! |
|--------------------|
```

Oops! Notice that there is now a defect in the padding for the Chinese version of the card.
If you look closely at the vertical alignment of the Hànzì symbols you can see that they are slightly wider than the latin letters and numbers. Normally we would be displaying the characters in a GUI which would allow us to make pixel-level adjustments. We could play around with a scaling factor to adjust the text padding, but it is not a problem we will worry about here.

Another problem to note is that the symbols display correctly in the output pane of IntelliJ, but they won't display correctly in a Command Tool or PowerShell Window; instead you will see question marks (but the padding will be correct!)

We could fix this by loading the appropriate language pack in Windows, but that's another problem we won't deal with in this course.