Working with Selected Classes from the Java API

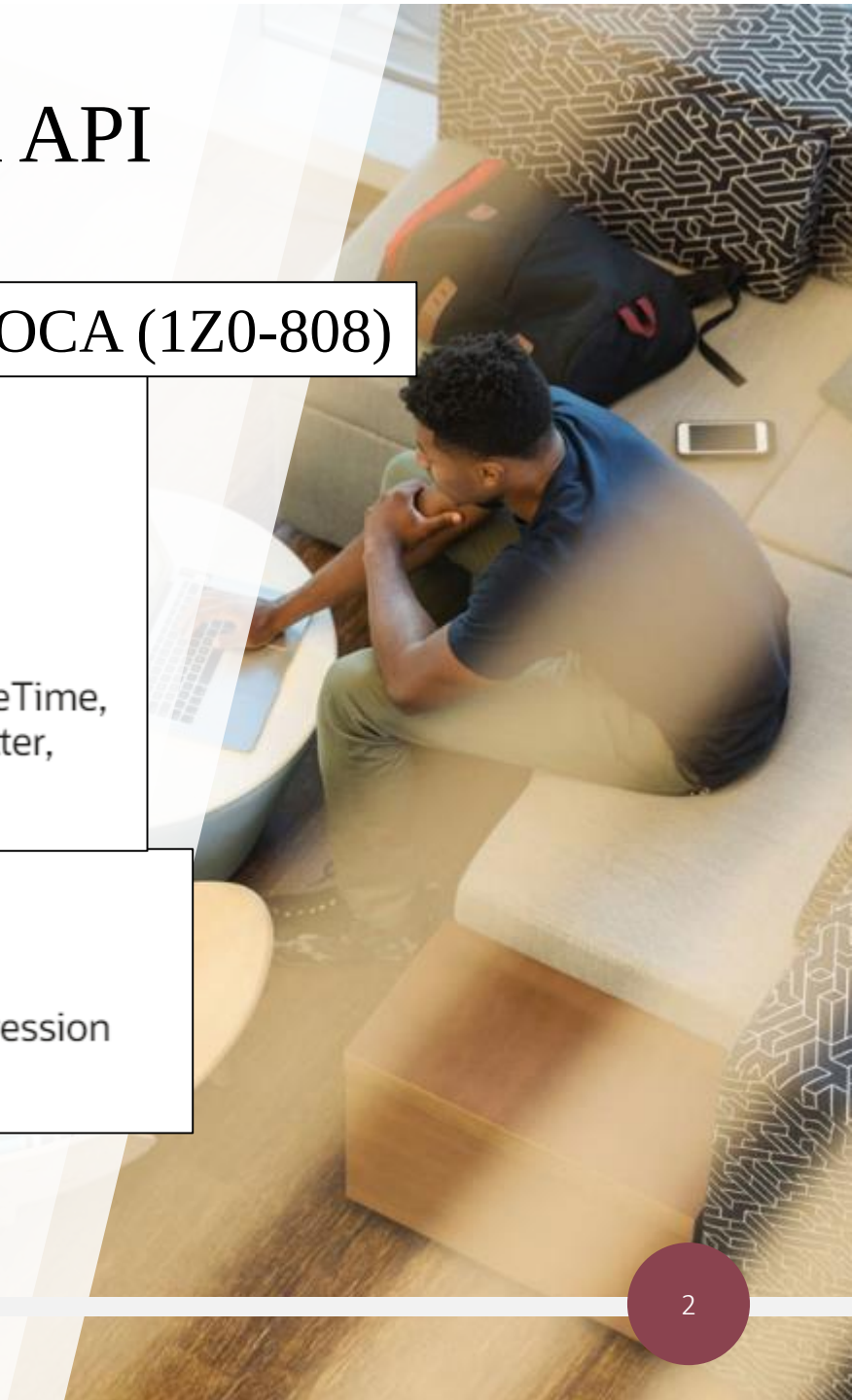
Working with Selected Classes from the Java API

Java 8 OCA (1Z0-808)

Working with Selected classes from the Java API

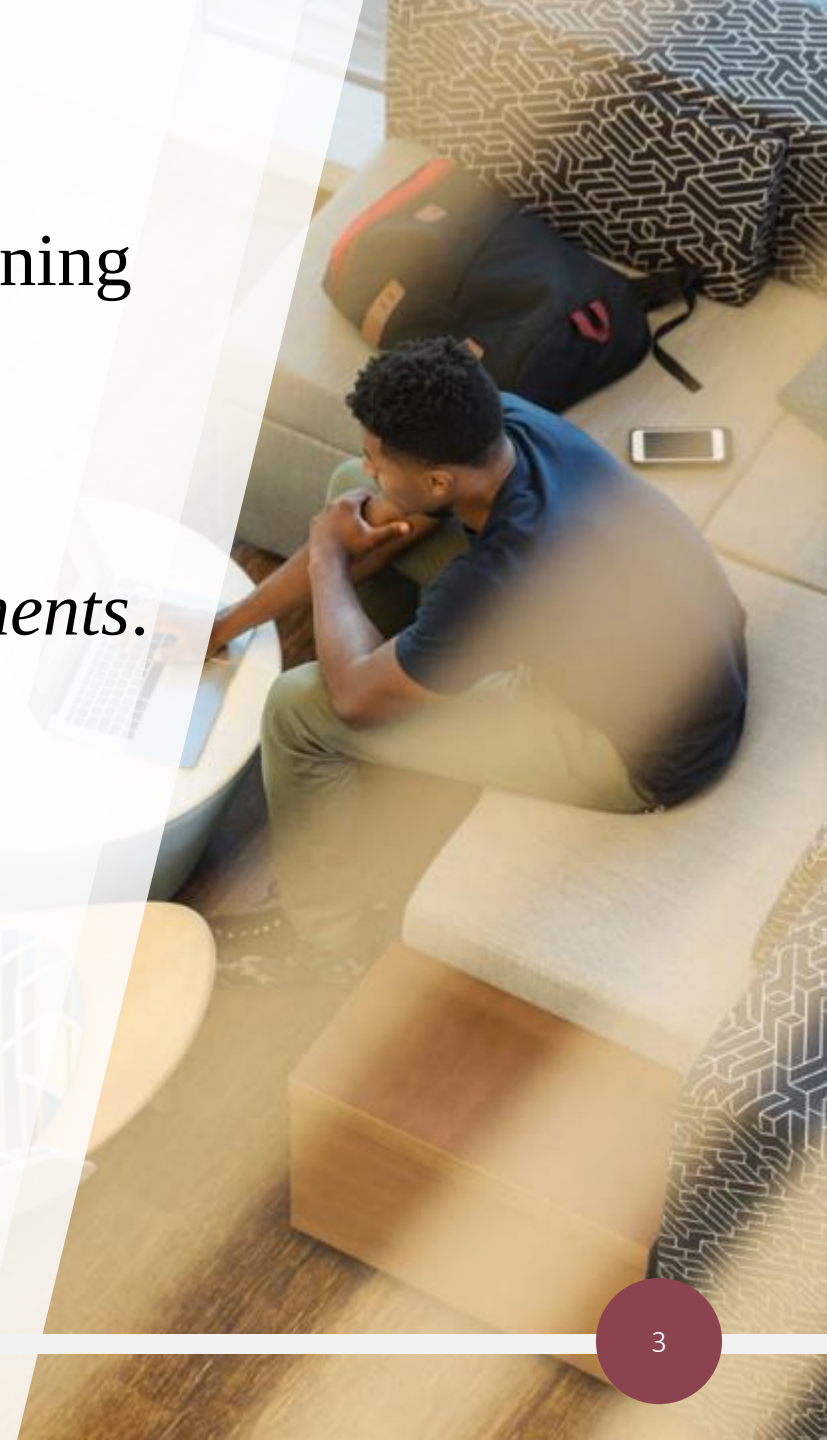
- ✓ Manipulate data using the StringBuilder class and its methods
- ✓ Create and manipulate Strings
- ✓ Create and manipulate calendar data using classes from java.time.LocalDateTime, java.time.LocalDate, java.time.LocalTime, java.time.format.DateTimeFormatter, java.time.Period

- ✓ Declare and use an ArrayList of a given type
- → Write a simple Lambda expression that consumes a Lambda Predicate expression



Interfaces

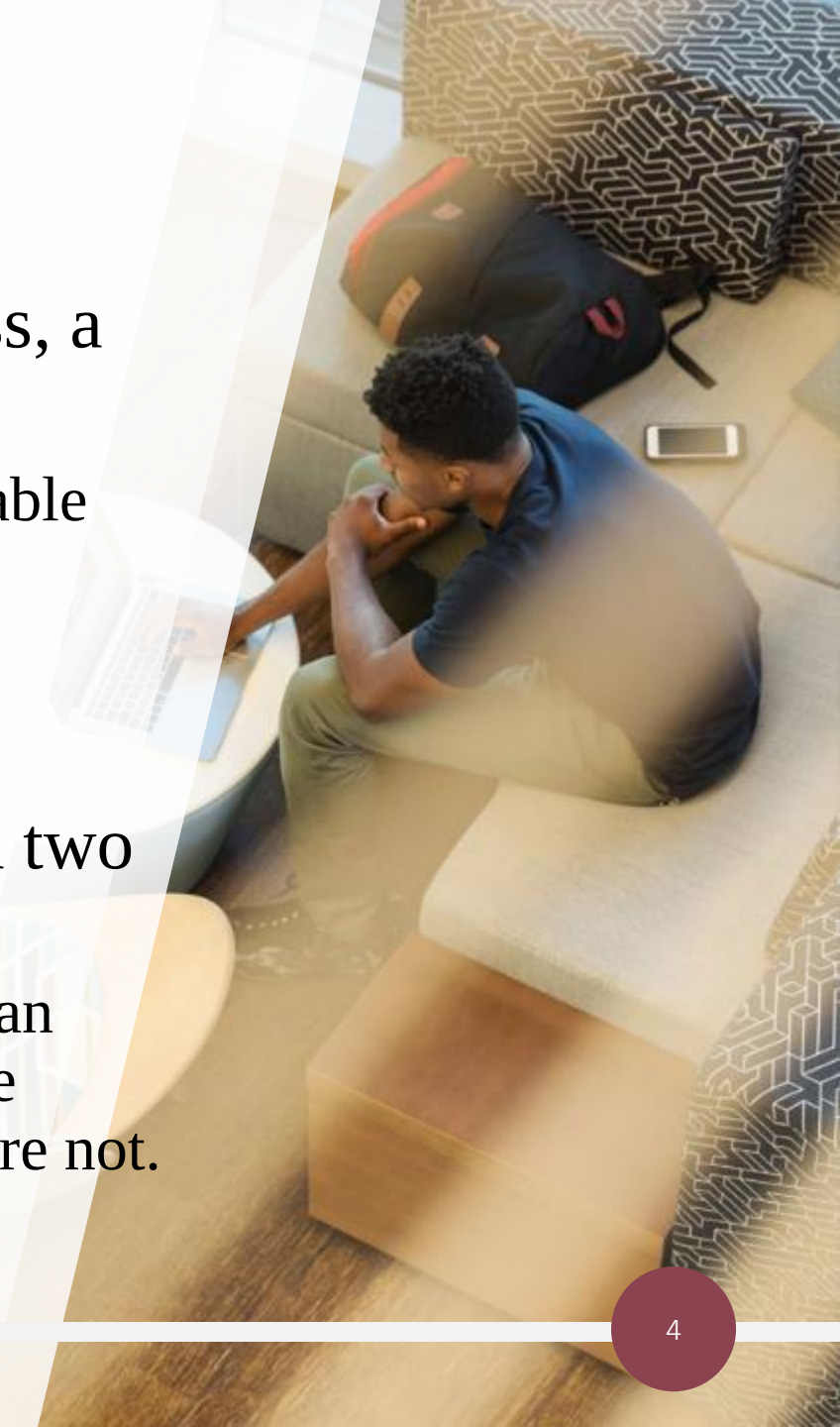
- In general, when you create an interface, you are defining a contract for *what* a class can do; without saying anything about *how* the class will do it.
- A class “signs” the contract with the keyword *implements*.
- When implementing an interface, you are agreeing to adhere (obey) to the contract defined in the interface.
- If a concrete (non-abstract) class is implementing an interface, the compiler will ensure that the class has implementation code for each abstract method in the interface.



Interfaces

- Whereas a class can extend from only one other class, a class can implement many interfaces.
  - class Dog extends Animal implements Moveable, Loveable
  - a Dog “is-a” : Animal, Moveable and Loveable

- As of Java 8, it is now possible to inherit **concrete** methods from interfaces. Interfaces can now contain two types of concrete methods: *static* and *default*.
  - Implementation classes are NOT required to implement an interface’s *static* or *default* methods. The *default* interface methods are inheritable but the *static* interface methods are not.



Interfaces

- Interfaces themselves are implicitly *abstract*
  - public abstract interface I{} ⇔ public interface I
  - top-level interfaces can have *public* or package-private access

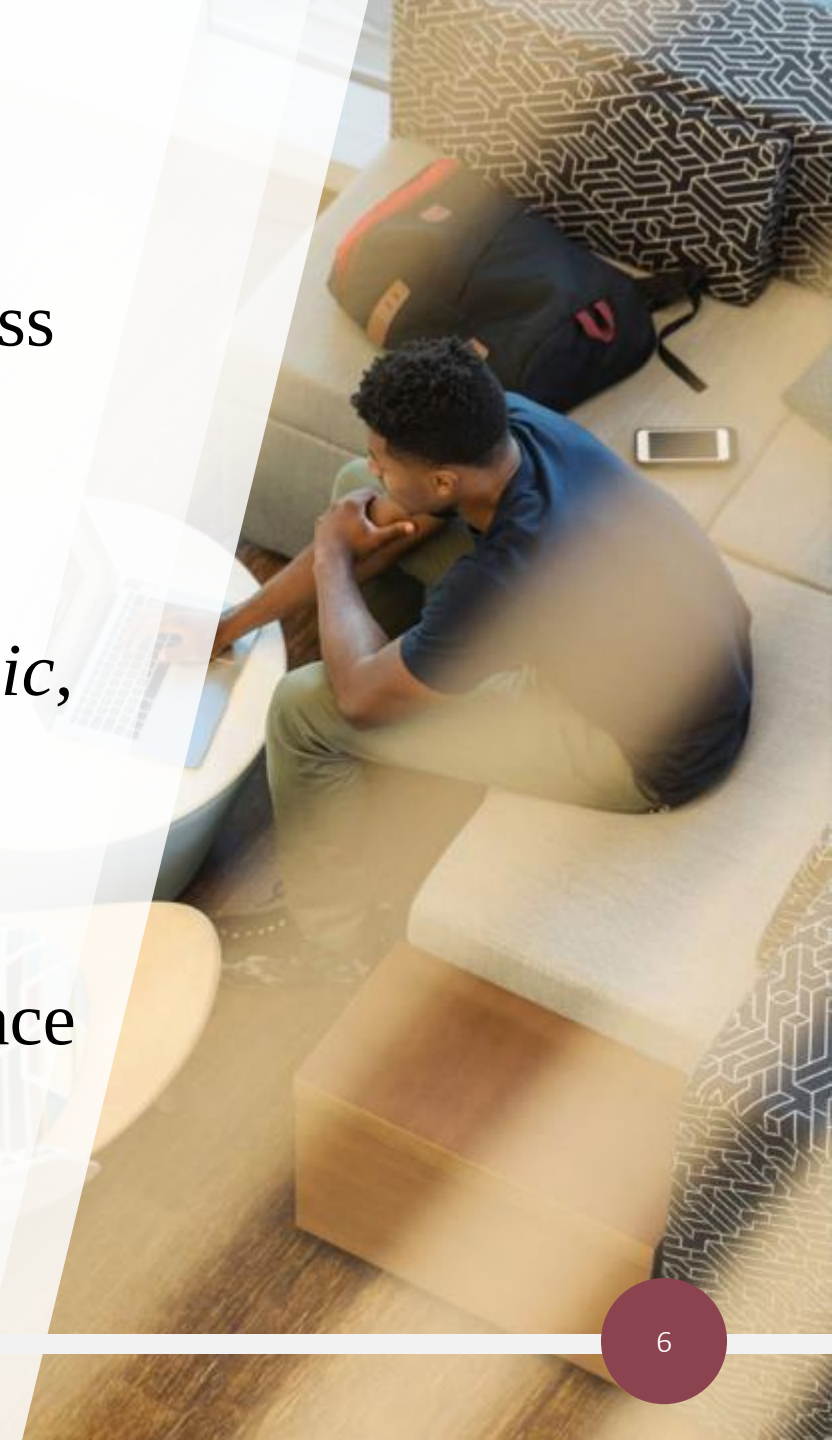
- All interface methods are implicitly *public*

Interfaces

- All interface methods are implicitly *abstract* (unless declared as *default* or *static*, the new features introduced in Java 8).

- All variables declared in an interface must be *public*, *static* and *final* i.e. interfaces can only declare constants (not instance variables).

- As with *abstract* classes you cannot *new* an interface type but they can be used as references:
  - *Printable p = new Printer(); // Printable is an interface*



Functional Interfaces

- A functional interface is an interface that has **only one abstract** method. This is known as the SAM (Single Abstract Method) rule.
  - *default* methods do not count
  - *static* methods do not count
  - methods inherited from *Object* do not count*

```
@FunctionalInterface
interface SampleFI{
    void m();
}
```

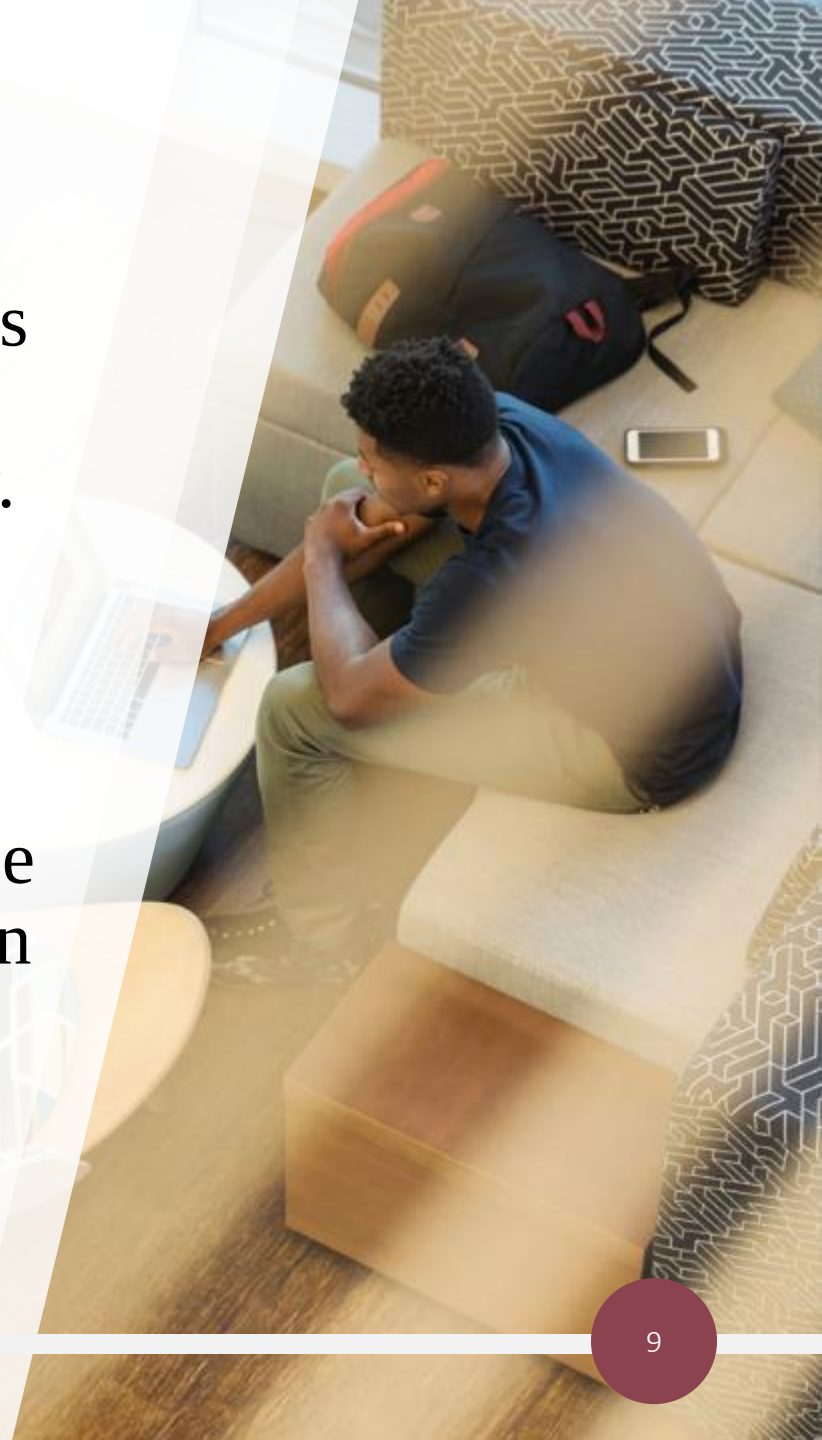
Lambdas

- A lambda expression is just a block of code that helps in making your code more concise.
- A lambda expression only works with functional interfaces.
- **A lambda expression is an instance of a class that implements a functional interface.**

Lambdas

- Lambdas look a lot like methods and in some quarters are called “anonymous methods”. However, it is an instance with everything but the method stripped away.

- A lot can be inferred (by the compiler) from the interface definition (which remember, has only one abstract method). The lambda expression is the instance that implements the interface that has been boiled down to the bare essentials.



```java
interface I{
    void m();// a functional interface i.e. it has only one
             // abstract method
}

public class BasicLambdas {
    public static void main(String[] args) {
        // pre-Java 8
        I i = new I(){
            @Override
            public void m(){
                System.out.println("I::m()");
            }
        };
        i.m();  //  I::m()

        // Java 8 - Lambda expression
        I lambdaI = () ->  {
            System.out.println("Lambda version");
        };
        I lambdaI2 = () ->  System.out.println("Lambda version");
        lambdaI.m();   //  Lambda version
        lambdaI2.m();  //  Lambda version
```

Method code.