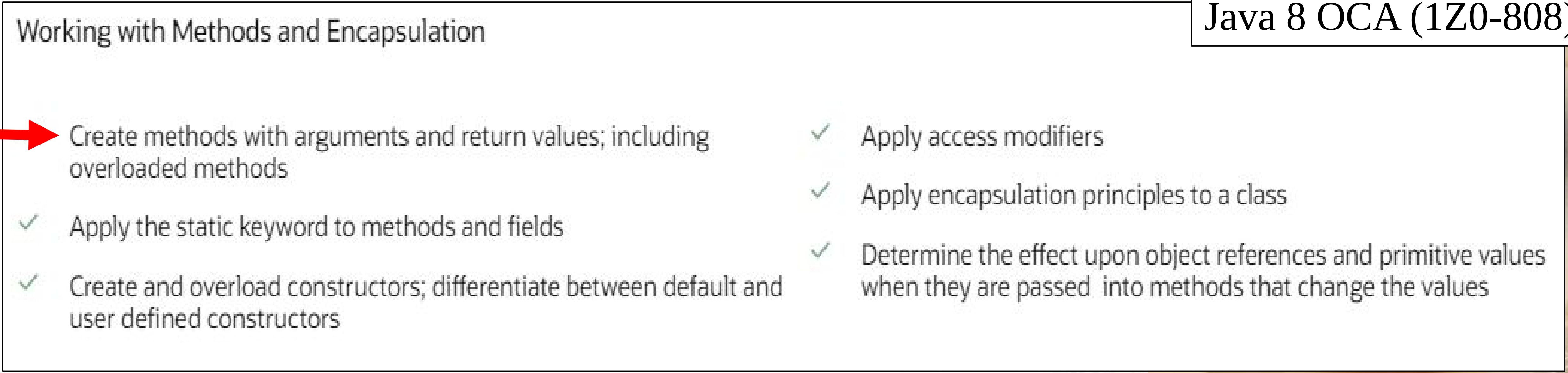
Working with Methods and Encapsulation

Working with Methods and Encapsulation

Java 8 OCA (1Z0-808)



Working with Methods and Encapsulation

- Create methods with arguments and return values; including overloaded methods
- ✓ Apply the static keyword to methods and fields
- ✓ Create and overload constructors; differentiate between default and user defined constructors
- ✓ Apply access modifiers
- ✓ Apply encapsulation principles to a class
- ✓ Determine the effect upon object references and primitive values when they are passed into methods that change the values

Methods

- Methods are a group (or block) of Java statements that are given a name for ease of reference.

- Very useful is a certain piece of code is going to be executed several times – rather than copy and paste; use a method.

- Once created, a method can then be invoked as often as needed.

- Methods can take in inputs and return a result.

Methods

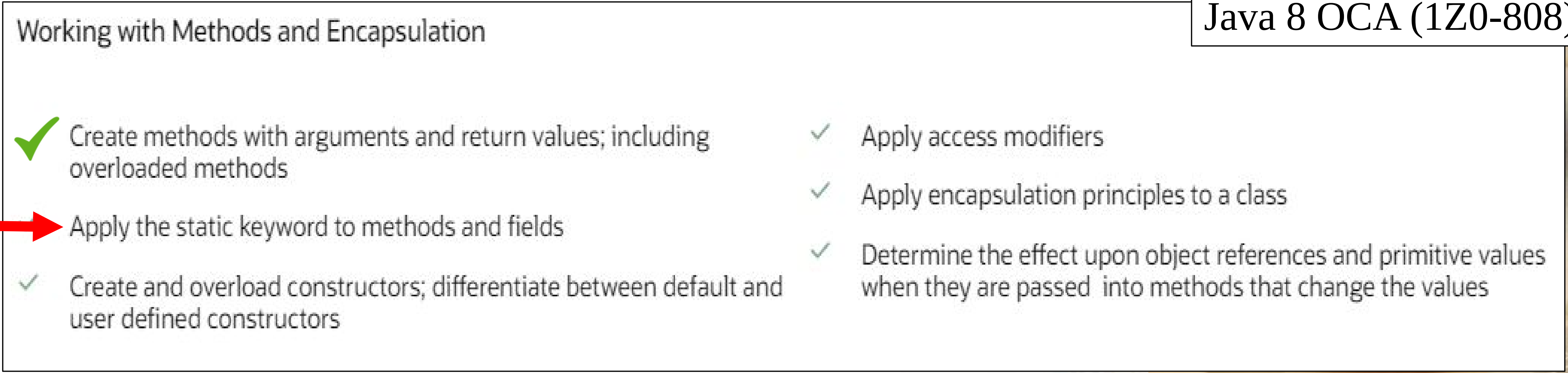
- A method has a “signature” which consists of the method name and the parameter type(s) and their order.
  - ➢ Note: the return type is **<u>not</u>** part of the method signature.

- A method’s code does not execute until it is called/invoked.
  a) define the method
  b) call the method

- When calling the method, be sure to pass down the arguments in the correct order.

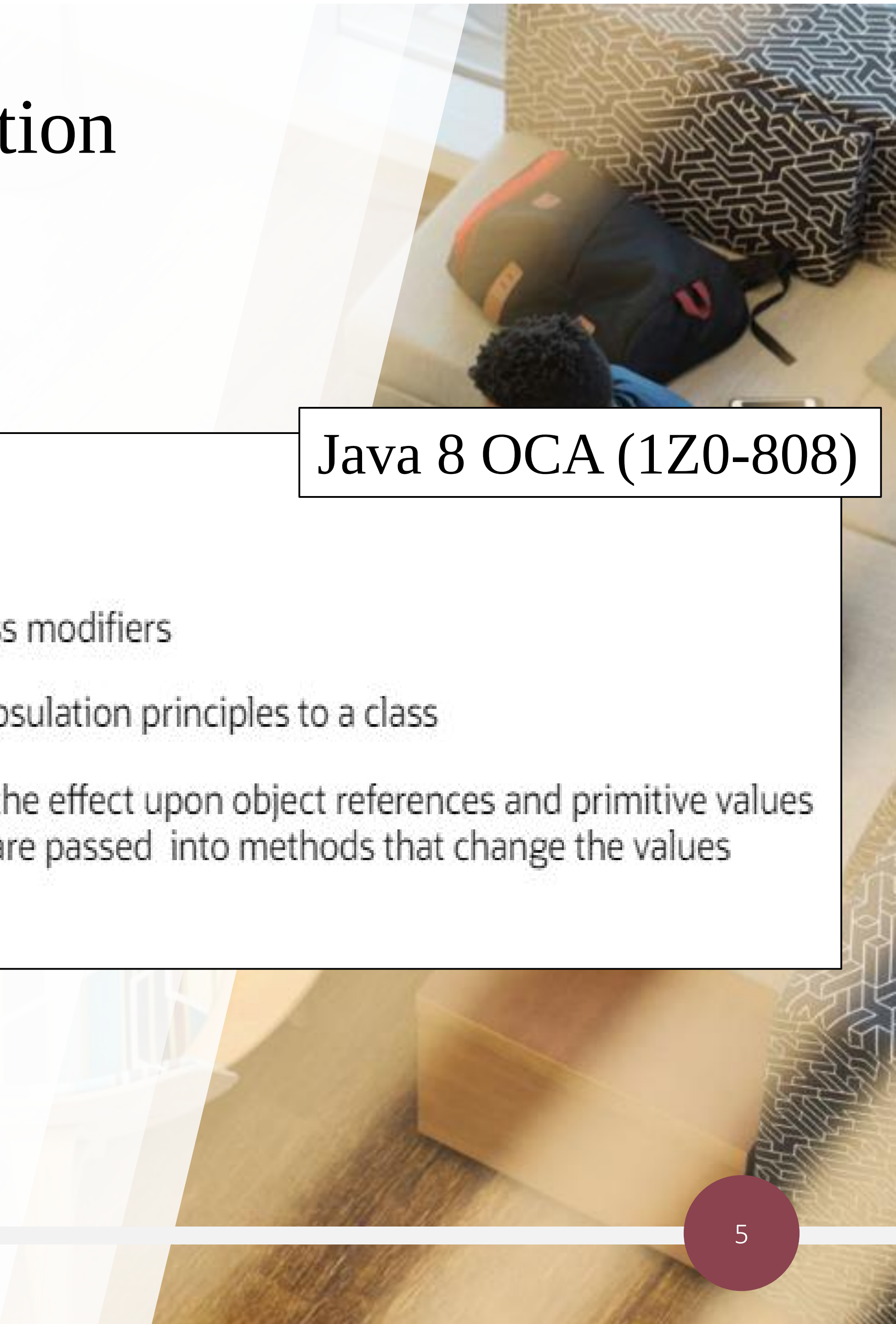
Working with Methods and Encapsulation

Java 8 OCA (1Z0-808)



Working with Methods and Encapsulation

- ✓ Create methods with arguments and return values; including overloaded methods
- Apply the static keyword to methods and fields
- ✓ Create and overload constructors; differentiate between default and user defined constructors
- ✓ Apply access modifiers
- ✓ Apply encapsulation principles to a class
- ✓ Determine the effect upon object references and primitive values when they are passed into methods that change the values



static keyword

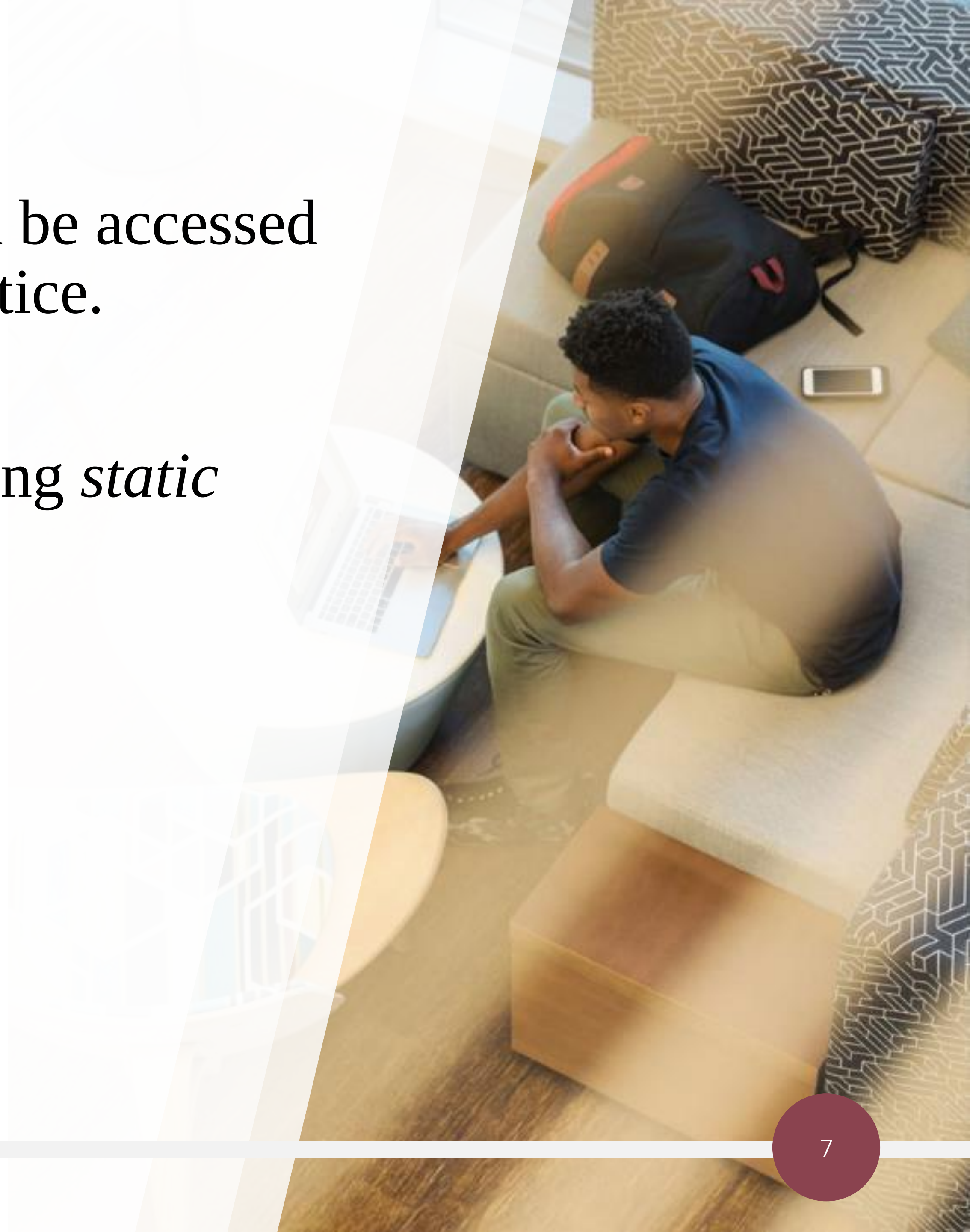
- There are situations where
    a) you don’t want every object to have it’s own copy of a variable i.e. you want the one copy to be shared by all instances
    b) you want to access or provide a utility method without the cost of creating an object

- The ‘static’ keyword is designed for both of these situations.
    a) class variables are shared among all instances
    b) utility methods do not require an object instance

static keyword

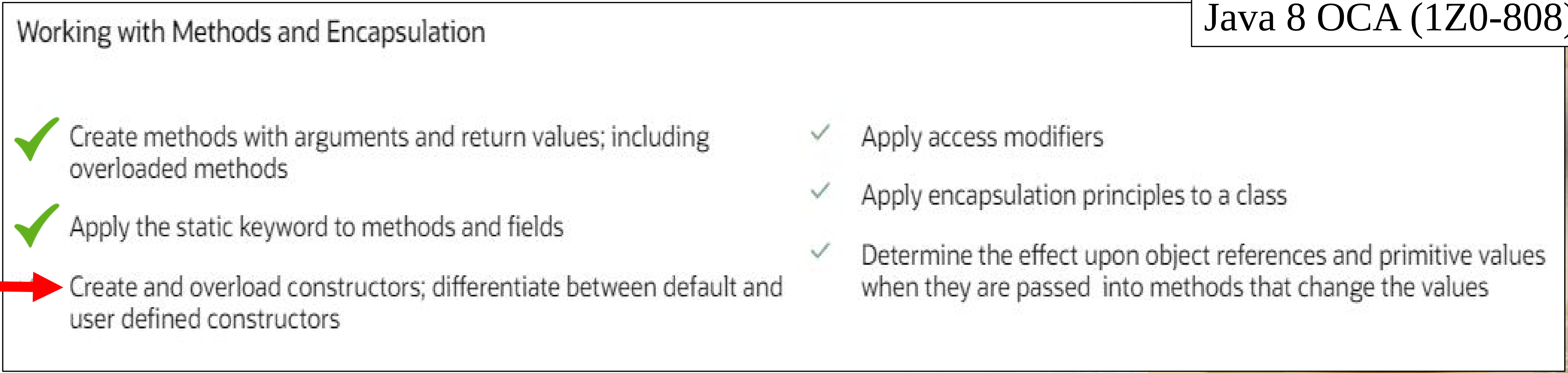
- While static members (data or methods) can be accessed using an object instance, this is not good practice.

- Use *ClassName.staticMember* when accessing *static* members.



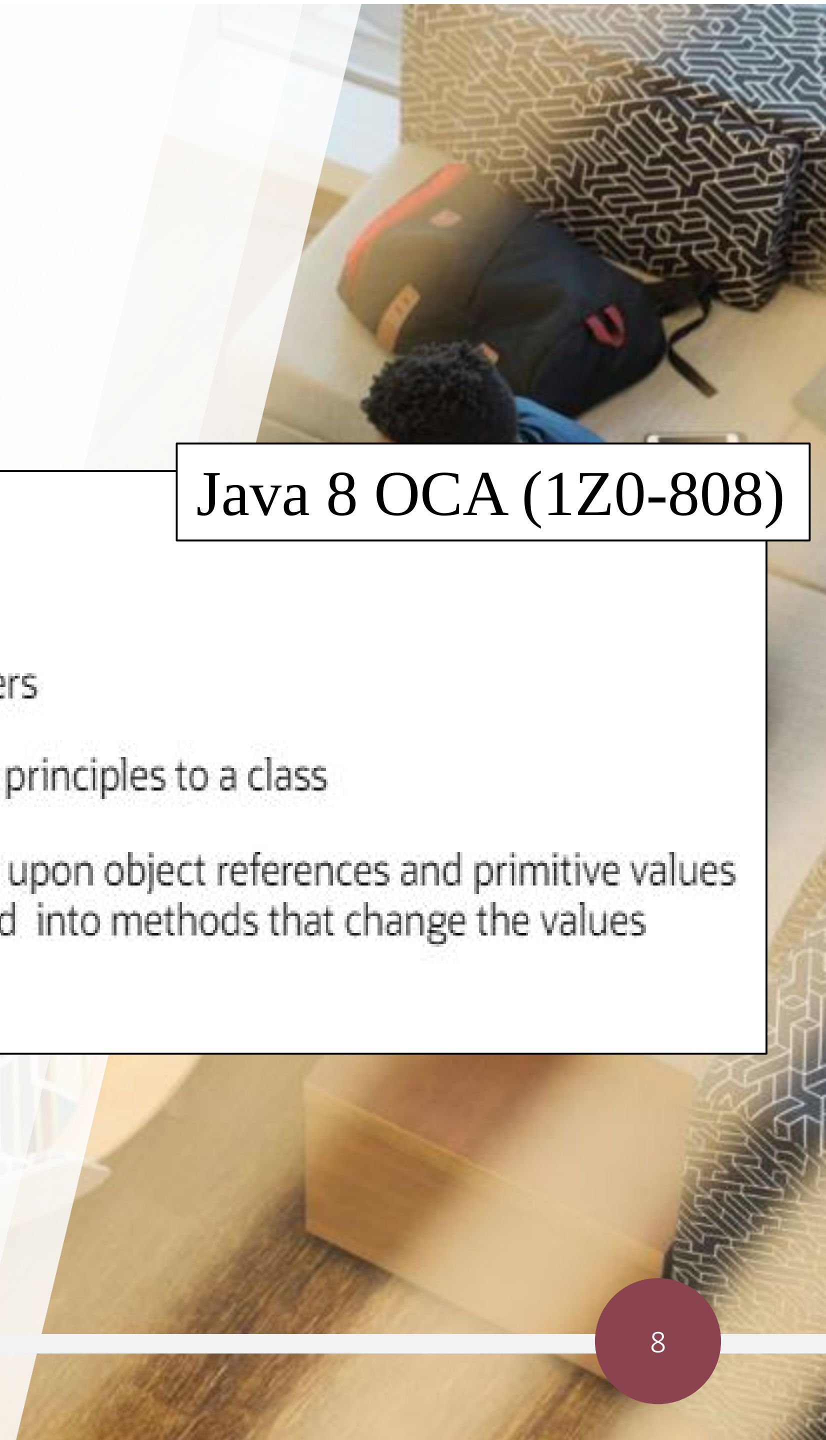
Working with Methods and Encapsulation

Java 8 OCA (1Z0-808)



Working with Methods and Encapsulation

- ✓ Create methods with arguments and return values; including overloaded methods
- ✓ Apply the static keyword to methods and fields
- → Create and overload constructors; differentiate between default and user defined constructors
- ✓ Apply access modifiers
- ✓ Apply encapsulation principles to a class
- ✓ Determine the effect upon object references and primitive values when they are passed into methods that change the values



Constructors

- A constructor is used to “construct” an object.

- A constructor is a special type of method – it has two identifying characteristics:
    a) same name as the class
    b) no return type – a reference to the newly constructed object is returned secretly in the background

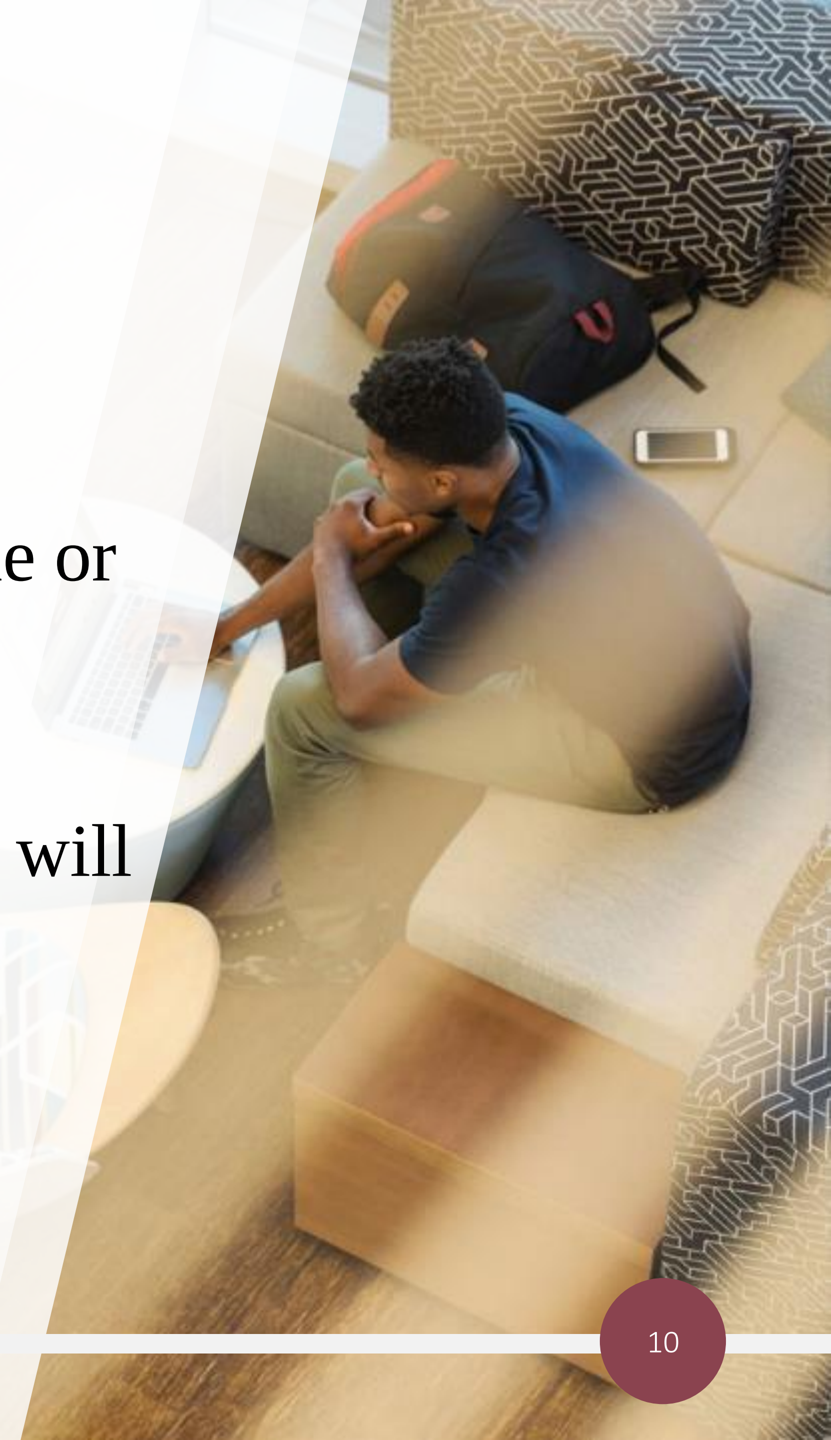
- The constructor is executed via “new”.

Constructors

- As with other methods, a constructor can also be overloaded.

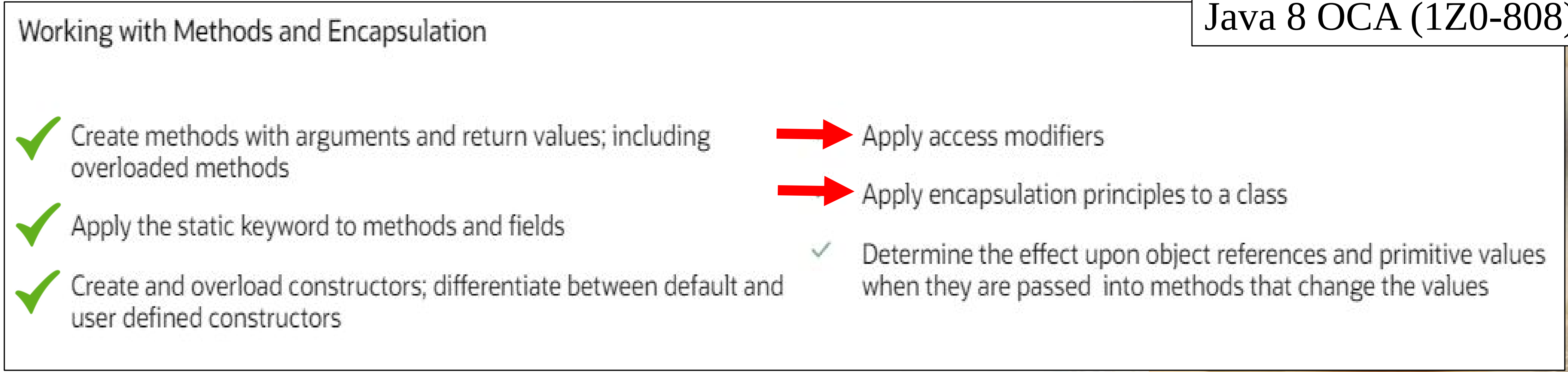
- Every class gets a constructor, whether you specify one or not.

- If you do not provide ANY constructors, a default one will be provided by the compiler:
    a) same access as the class
    b) invokes “super();”



Working with Methods and Encapsulation

Java 8 OCA (1Z0-808)



Working with Methods and Encapsulation

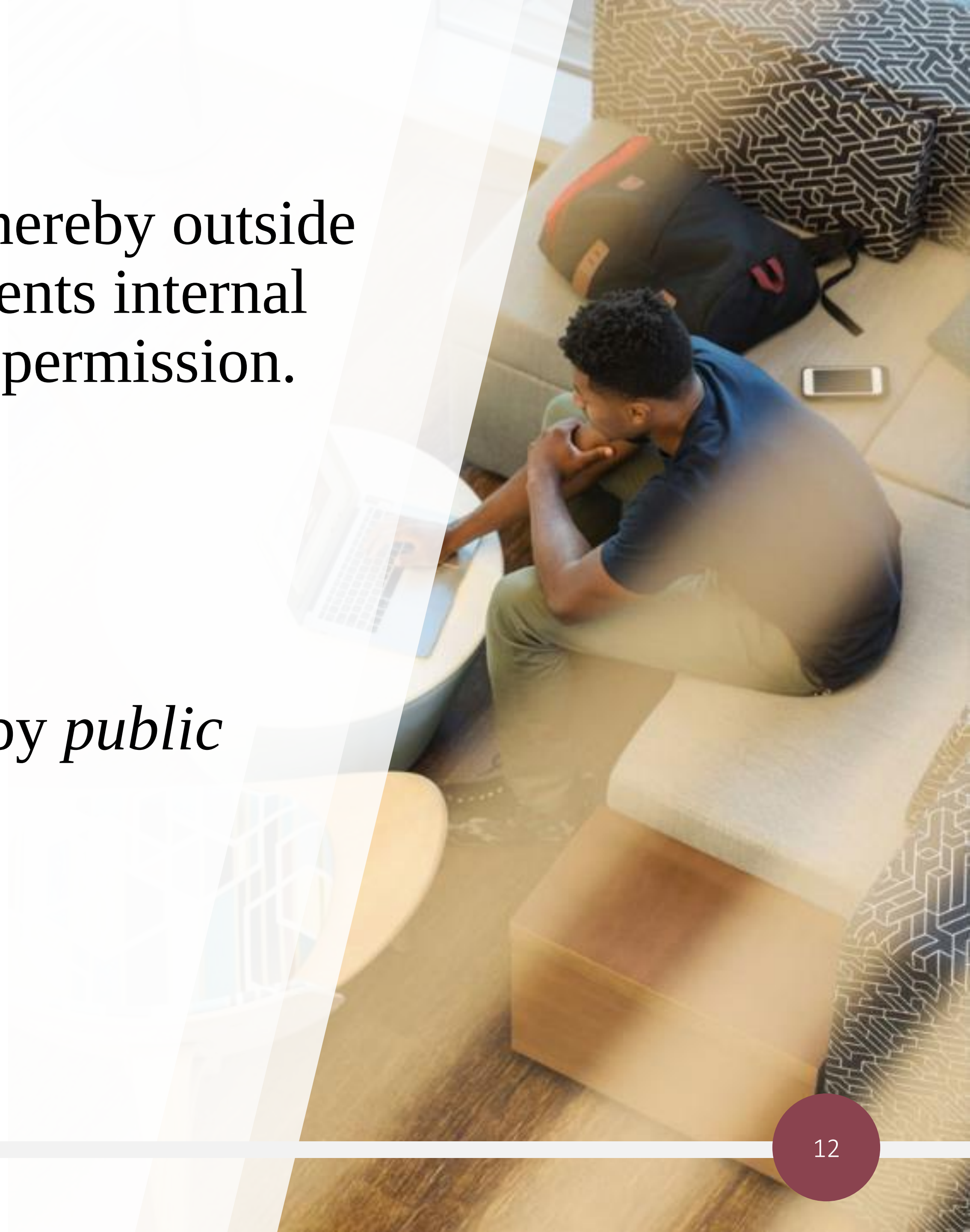
- ✓ Create methods with arguments and return values; including overloaded methods
- ✓ Apply the static keyword to methods and fields
- ✓ Create and overload constructors; differentiate between default and user defined constructors
- Apply access modifiers
- Apply encapsulation principles to a class
- ✓ Determine the effect upon object references and primitive values when they are passed into methods that change the values

Encapsulation

- Encapsulation is a core principle of OOP whereby outside components cannot change/modify a components internal state without the components knowledge and permission.

- Often referred to as “data hiding”.

- Enabled in Java via *private* data accessible by *public* methods.



Access Modifiers

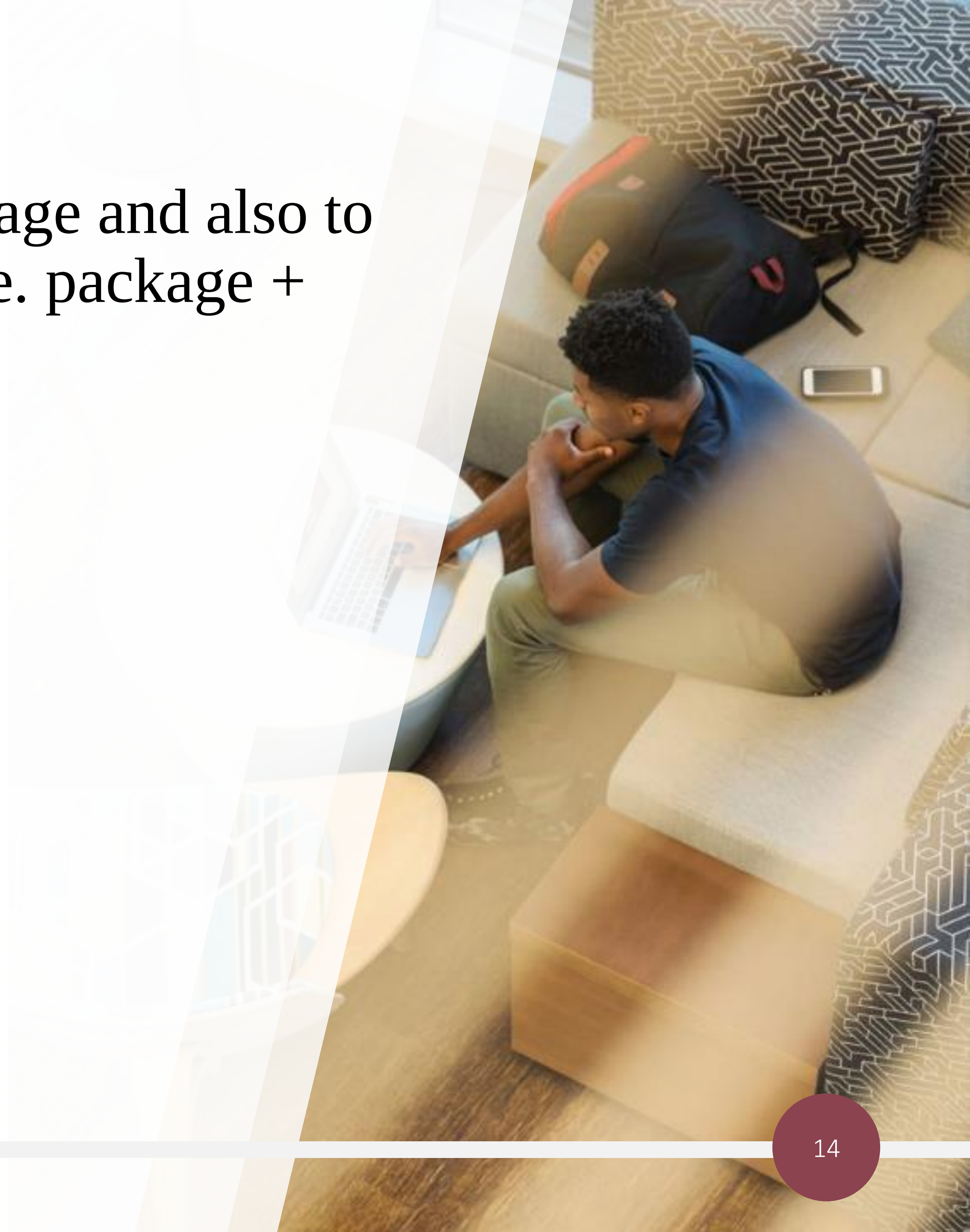
- Java provides the following access modifiers (in order of most restrictive to least restrictive):
  - private
  - package-private
  - protected
  - public

- *private* – accessible only to the class in which the member is defined
- package-private – no keyword applies this access; accessible to the class and any other class in the same package

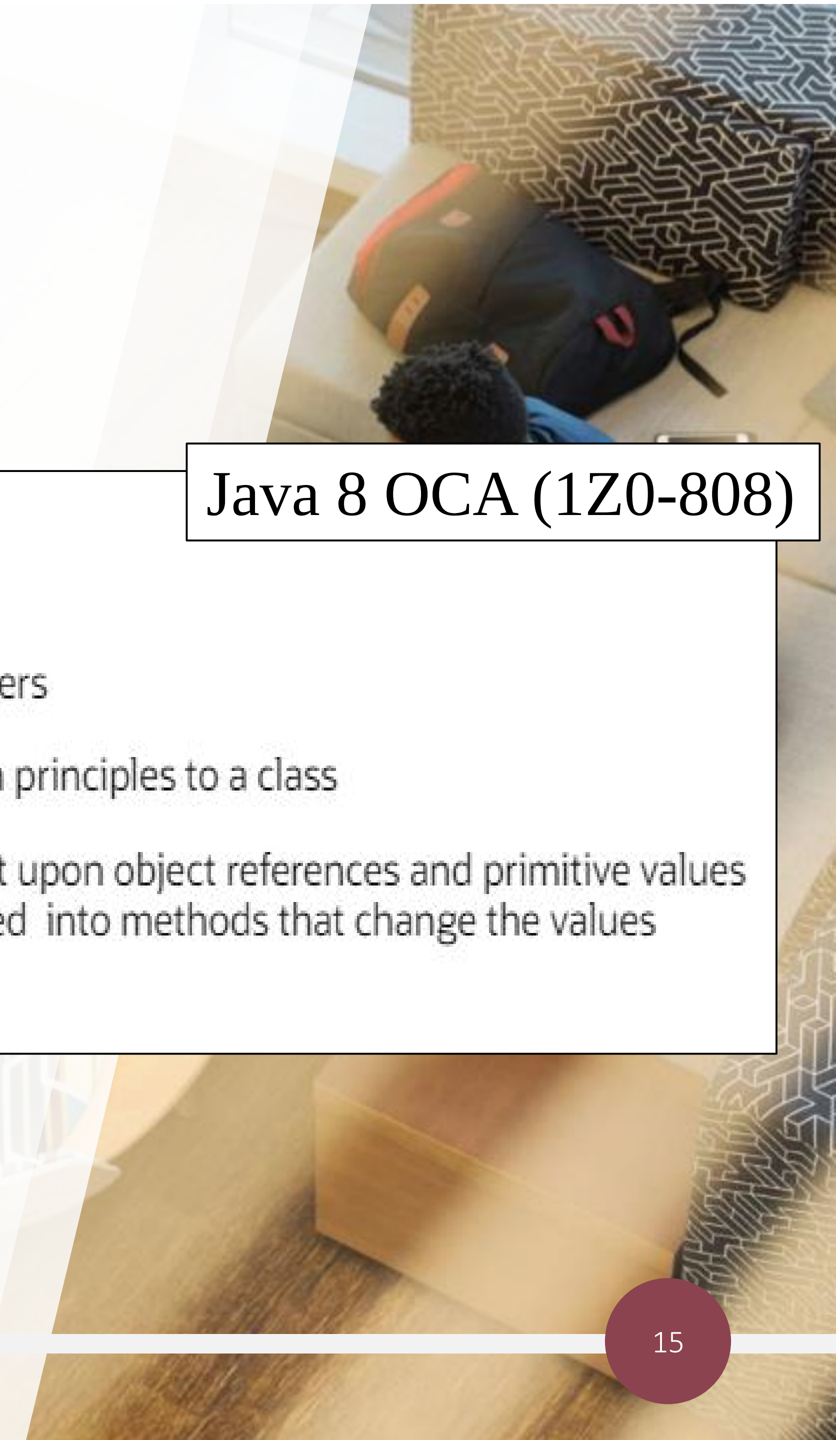
Access Modifiers

- *protected* – accessible within the same package and also to children of the class outside of the package i.e. package + children.

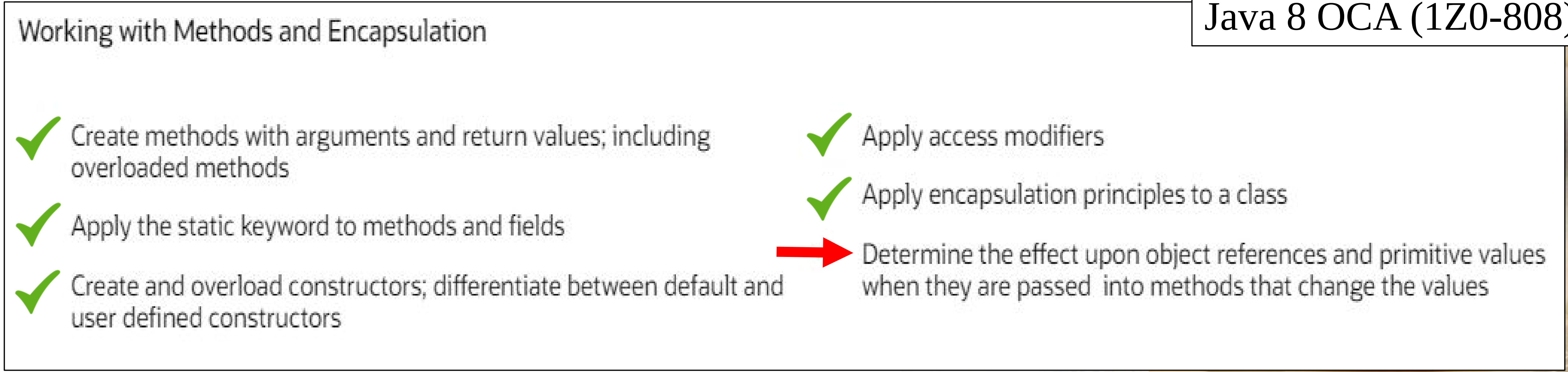
- *public* – available everywhere.



Working with Methods and Encapsulation



Java 8 OCA (1Z0-808)



Working with Methods and Encapsulation

- ✓ Create methods with arguments and return values; including overloaded methods
- ✓ Apply the static keyword to methods and fields
- ✓ Create and overload constructors; differentiate between default and user defined constructors
- ✓ Apply access modifiers
- ✓ Apply encapsulation principles to a class
- → Determine the effect upon object references and primitive values when they are passed into methods that change the values

Call-By-Value

- Java uses Call-By-Value
- A copy of the argument is passed to the method.
- However, there is a massive difference in the effect of passing a copy of a primitive and passing a copy of a reference.
  - primitive – the called method **cannot** change the primitive value in the caller method
  - reference – the called method **can** change the object (state) that the caller method is looking at