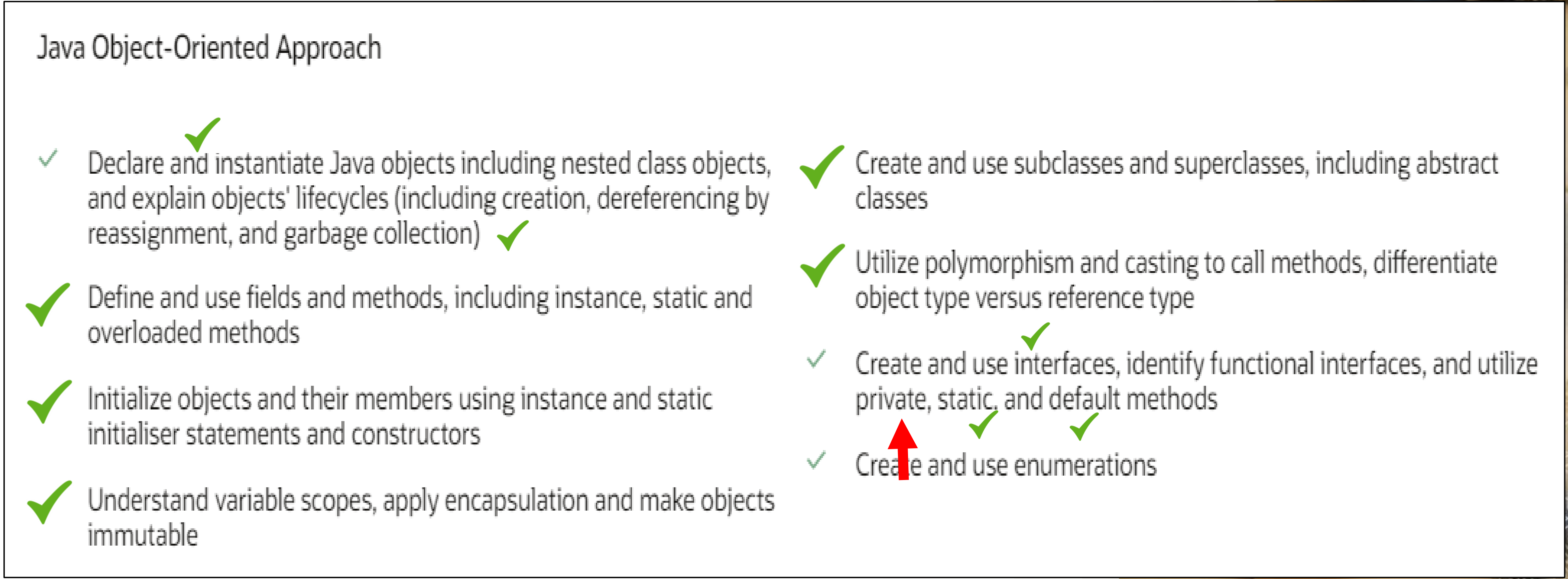
Java Object Oriented Approach

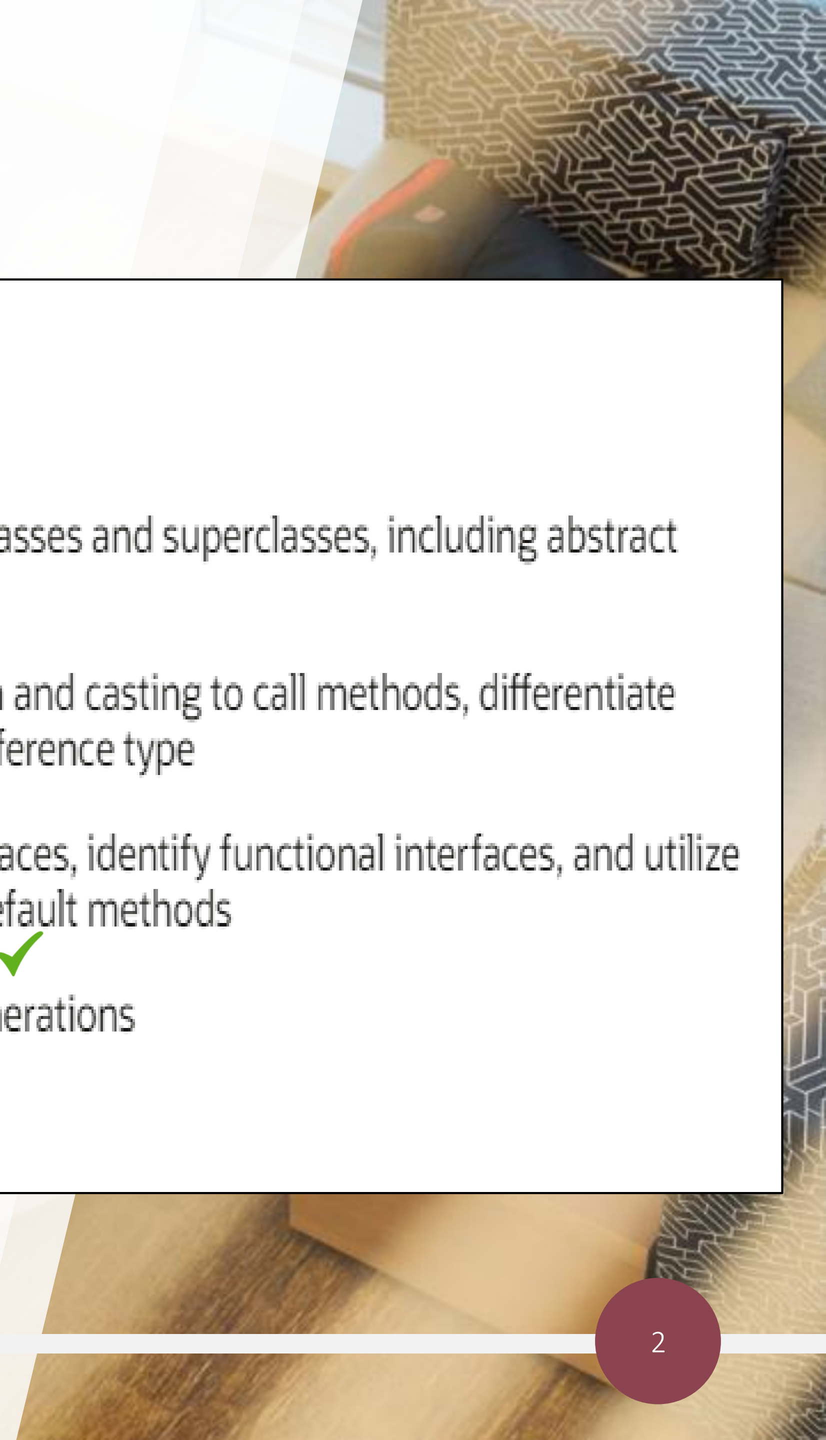
private interface methods

Java Object-Oriented Approach



Java Object-Oriented Approach

- ✓ Declare and instantiate Java objects including nested class objects, and explain objects' lifecycles (including creation, dereferencing by reassignment, and garbage collection) ✓
- ✓ Define and use fields and methods, including instance, static and overloaded methods
- ✓ Initialize objects and their members using instance and static initialiser statements and constructors
- ✓ Understand variable scopes, apply encapsulation and make objects immutable
- ✓ Create and use subclasses and superclasses, including abstract classes
- ✓ Utilize polymorphism and casting to call methods, differentiate object type versus reference type
- ✓ Create and use interfaces, identify functional interfaces, and utilize private, static, and default methods ✓ ✓
- ✓ Create and use enumerations

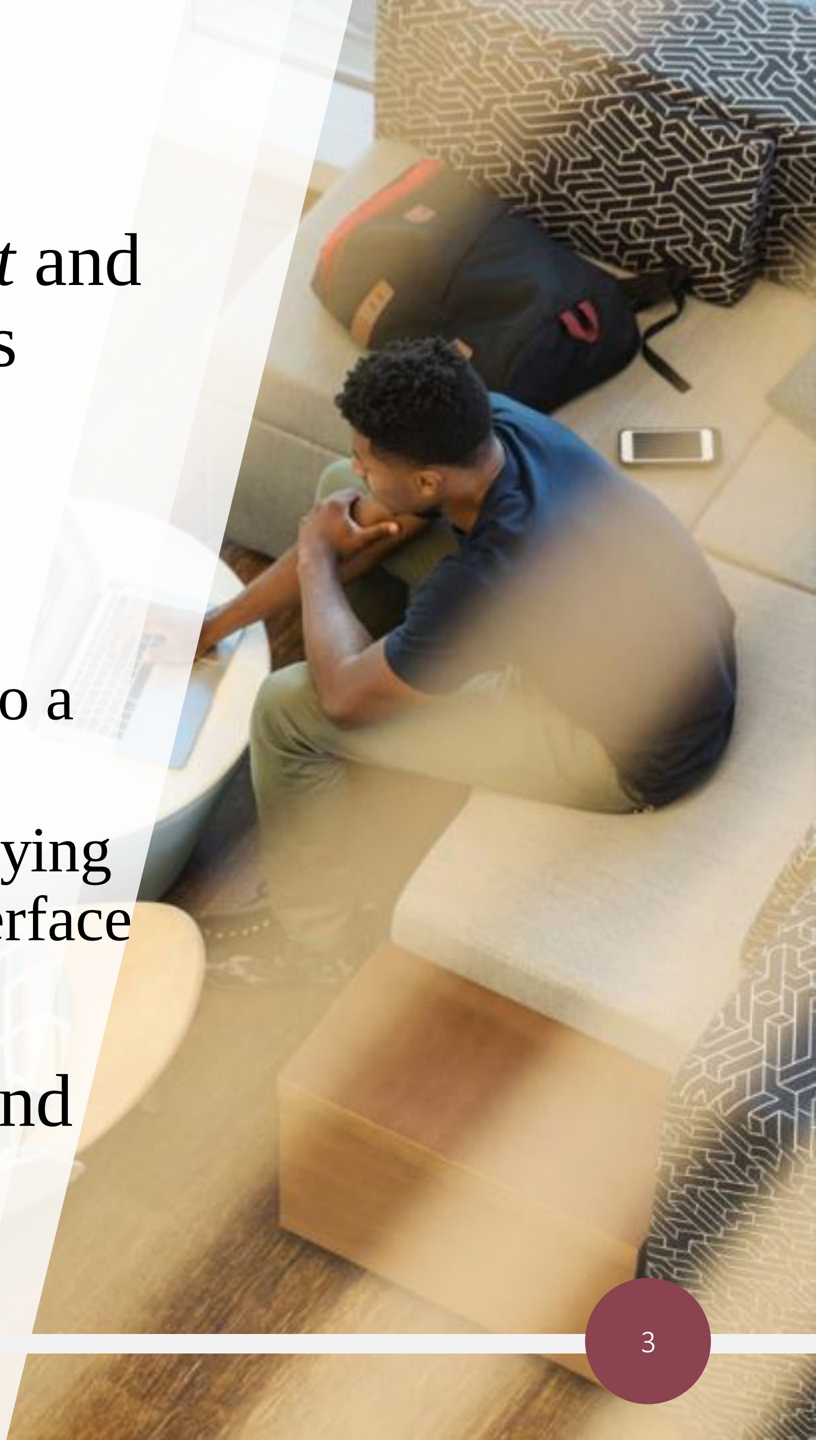


private interface methods

- As we have seen, interfaces can have *abstract*, *default* and *static* methods. In addition, *default* and *static* methods have implementations.

- Introduced in Java 9, *private* interface methods:
  a) reduces code duplication - duplicated code can be put into a *private* interface method
  b) improves encapsulation in interfaces - parts of the underlying implementation can now be hidden from users of the interface

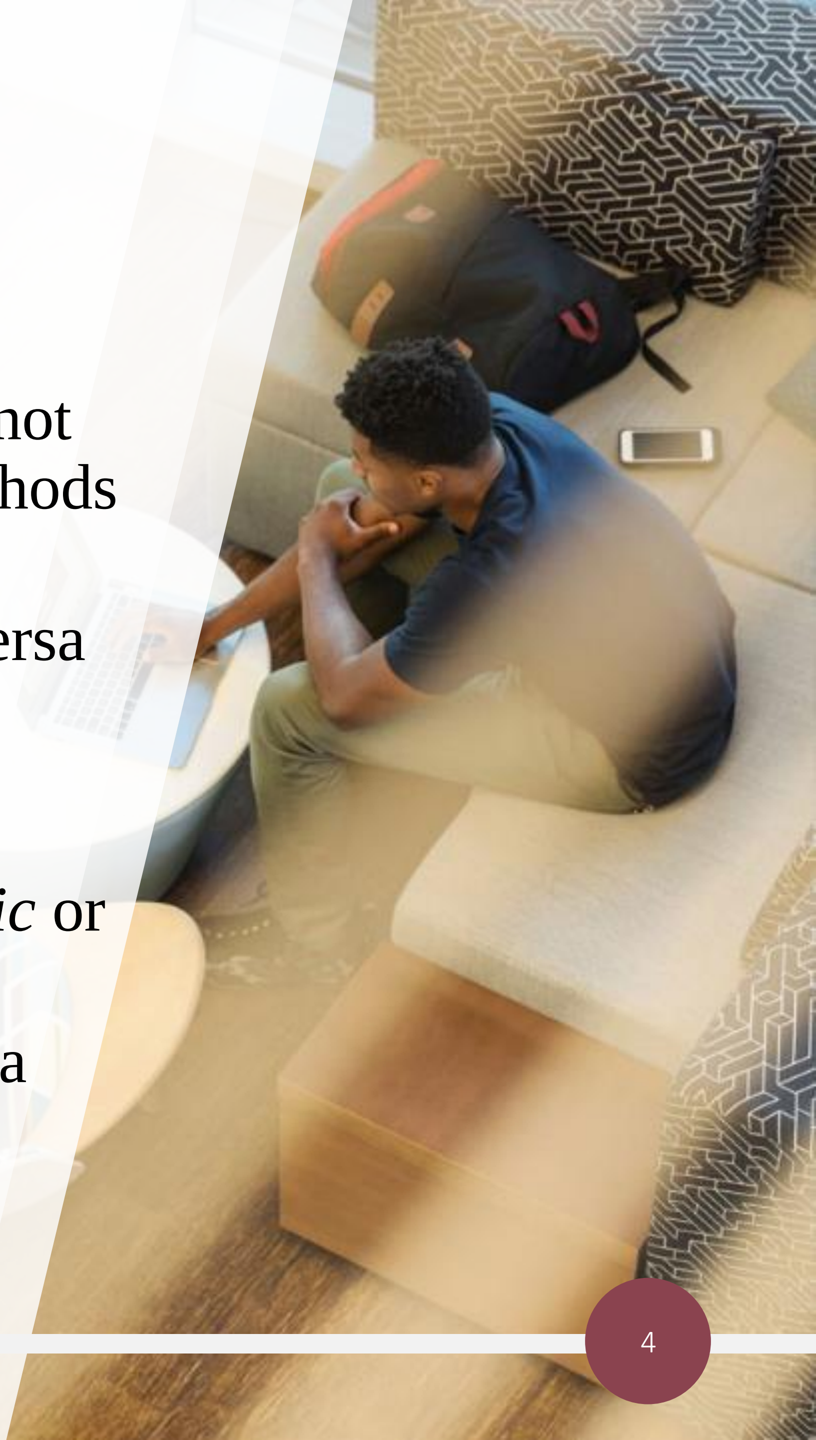
- *private* interface methods can be *static* or non-static and have an implementation so cannot be *abstract*.



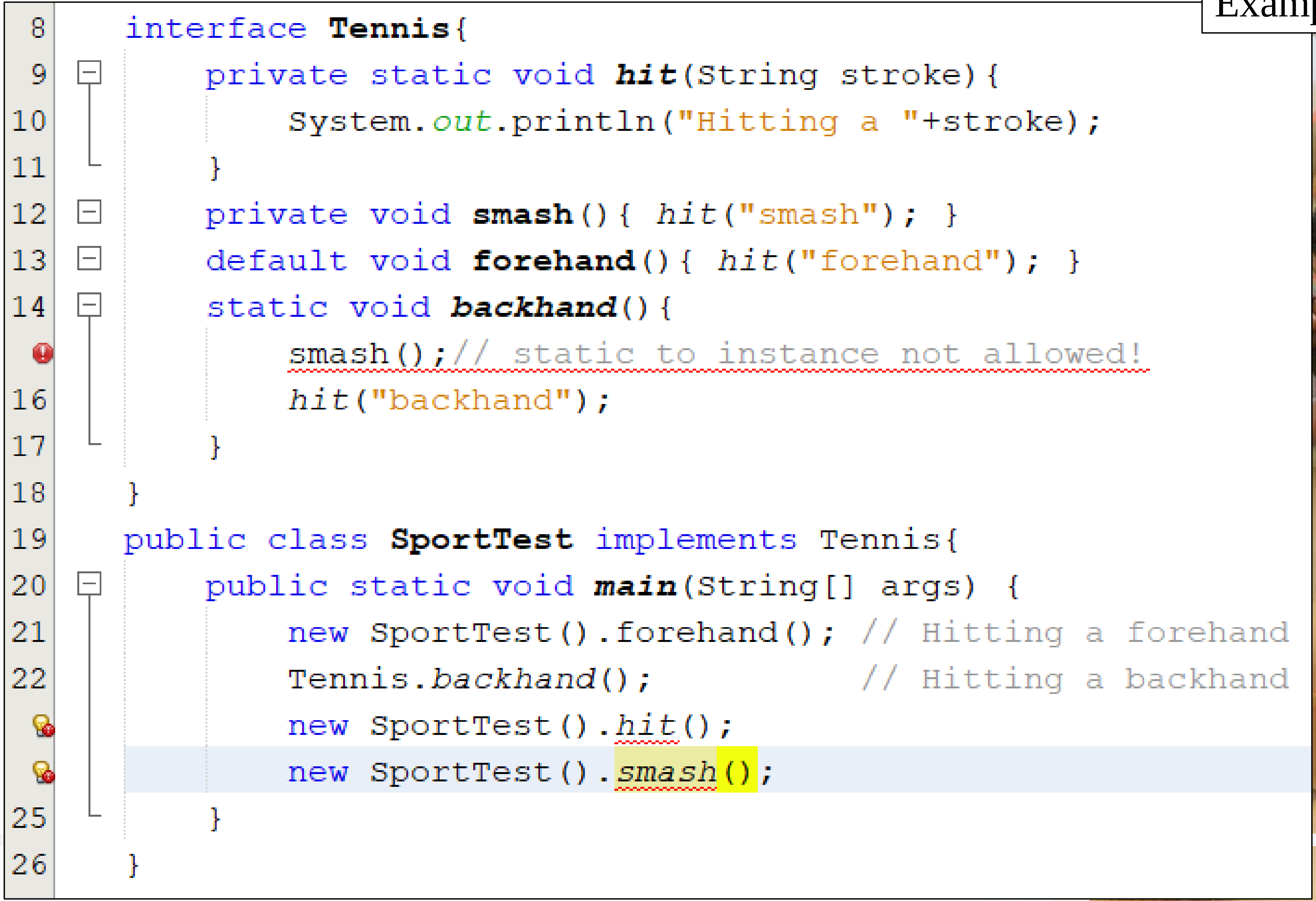
private interface methods

- *private* method (non-static)
  - you can access a *private* method from a *default* or *private* method; but not from a *static* method. Remember, you cannot call an instance method from a *static* context (as *static* methods have no *this* reference).
  - as with classes, instance to static (I.S.) is ok but not vice versa

- *private static* method
  - You can access a *private static* method from a *default*, *static* or *private* method.
  - as with classes, instance to static (I.S.) ok but not vice versa



Example



```java
interface Tennis{
    private static void hit(String stroke){
        System.out.println("Hitting a "+stroke);
    }
    private void smash(){ hit("smash"); }
    default void forehand(){ hit("forehand"); }
    static void backhand(){
        smash();// static to instance not allowed!
        hit("backhand");
    }
}
public class SportTest implements Tennis{
    public static void main(String[] args) {
        new SportTest().forehand(); // Hitting a forehand
        Tennis.backhand();          // Hitting a backhand
        new SportTest().hit();
        new SportTest().smash();
    }
}
```