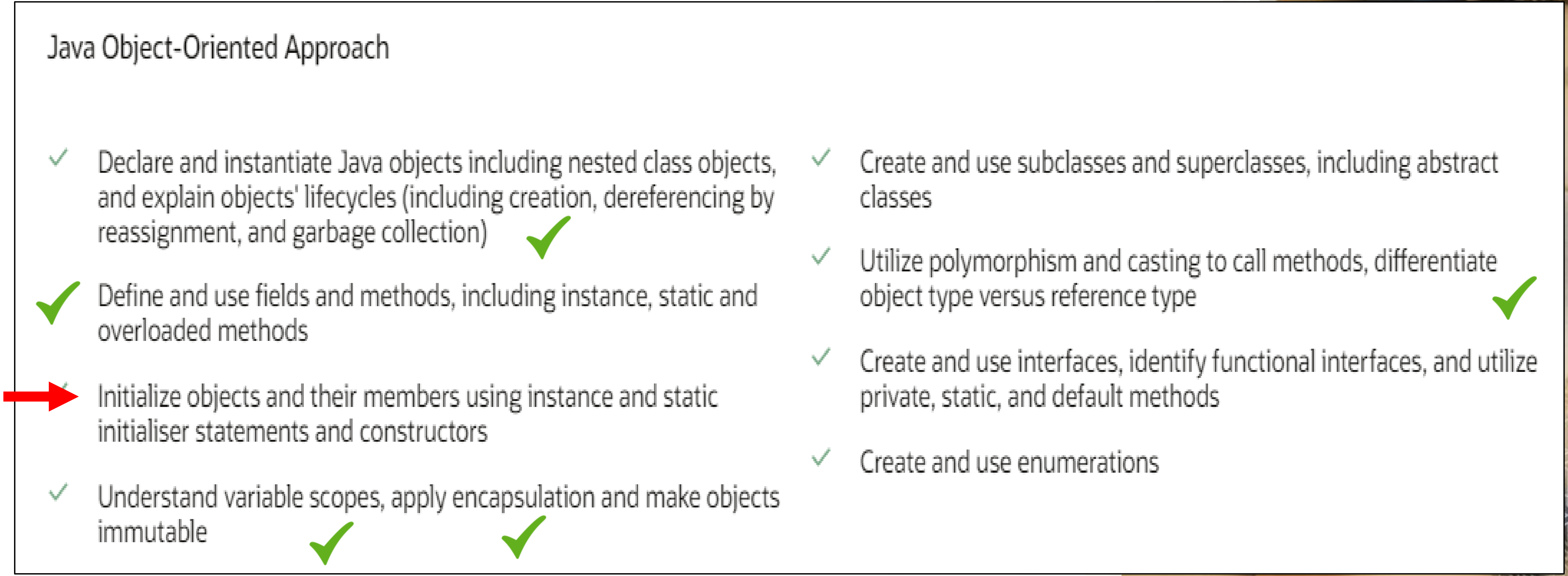
Java Object Oriented Approach

object initialisation

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

Object Initialisation

- Consider a class *Cow*:

1. The first time an object of type *Cow* is created OR the first time a *static* field or method in *Cow* is accessed, the JVM locates Cow.class.
2. As Cow.class is loaded, all of its *static* initialisers are run (in the order they appear in the code). This happens only once, as Cow.class is loaded for the first time.
3. When you create a new *Cow* i.e. *new Cow()*, space is allocated on the heap and it is wiped to zero. This sets all the primitives in that object to their default values i.e. 0 or 0.0 for numbers; *false* for *boolean* and *null* for references e.g. *Integer*, *String* etc..

Object Initialisation

4. All the non-static (instance) initialisation now takes place e.g. instance variables and instance blocks. As with static initialisation, the order they appear in the code determines the order they are initialized in.
5. Constructor(s) are executed.

Note: When we discuss Inheritance, extra steps are involved. Acronym “sic” helps – statics, instance, constructors.