# Java Object Oriented Approach

Enumerations

# Java Object-Oriented Approach

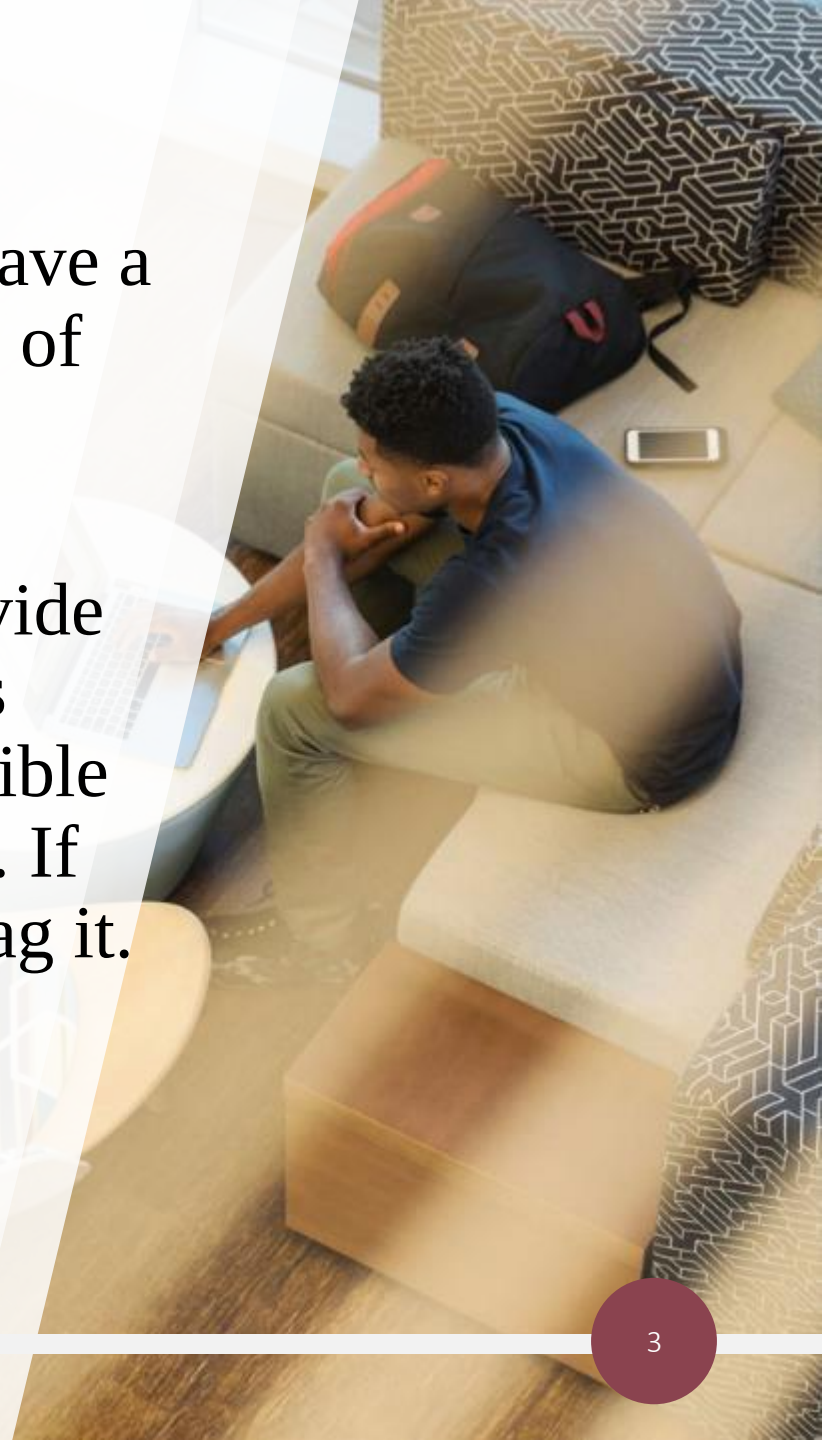
## Java Object-Oriented Approach

- 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

# Enumerations

- Enumerations (or enums) are very useful when you have a type that can have only a finite set of values e.g. days of the week, directions, seasons etc…

- Enumerations are special classes that allow us to provide a type-safe solution by restricting the object instances allowed. Using numeric or *String* constants, it is possible to use an incorrect value and only find out at runtime. If you use an incorrect enum value, the compiler will flag it.



# Enumerations

- The values of an enum are expressed similarly to constants i.e. in capital letters. However, these are in fact references to the only object instances of the enum (class) allowed.

- Here is an example:

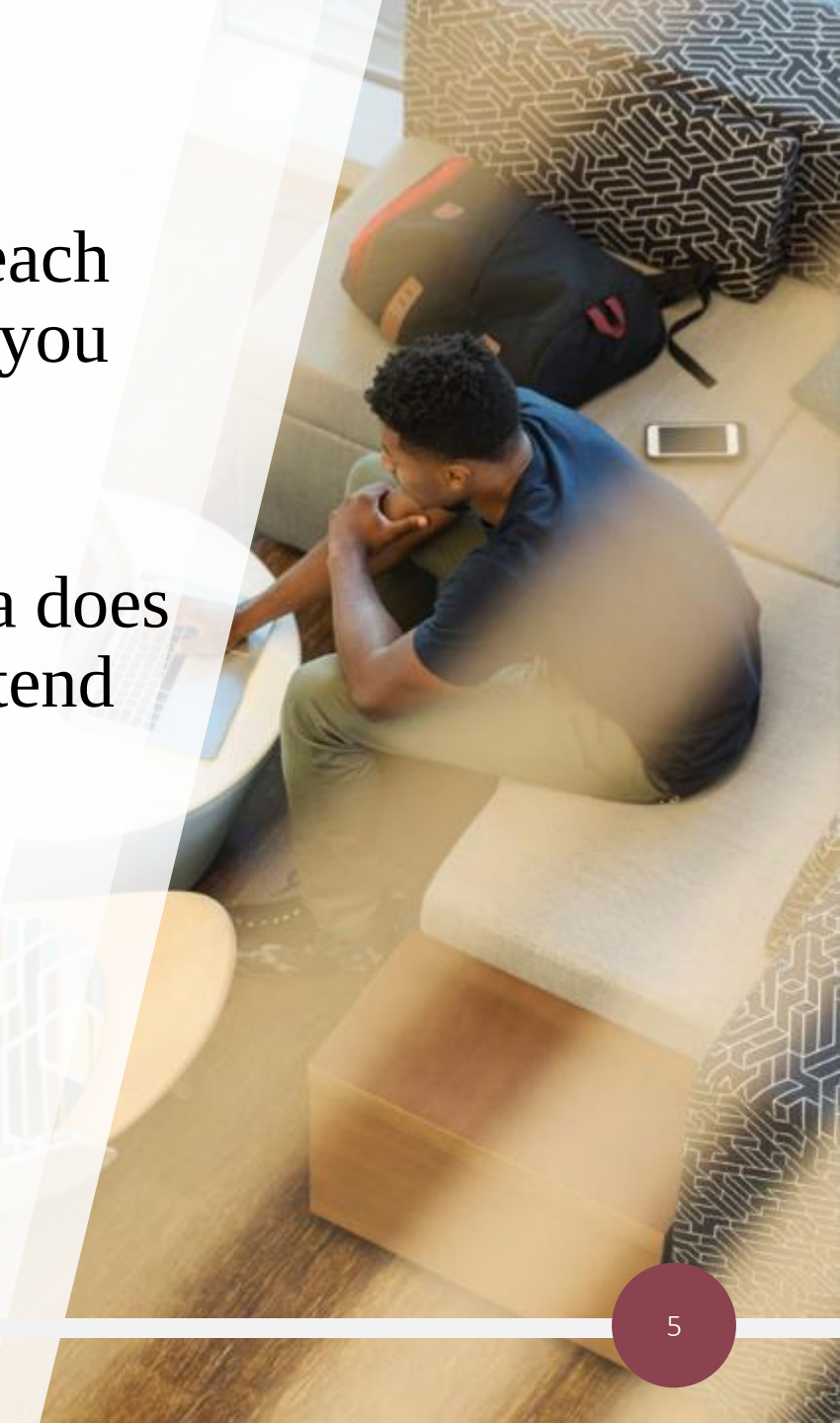
```
public enum Direction{
     NORTH, SOUTH, EAST, WEST
}
```

# Enumerations

- Enumerations are like *static* and *final* constants i.e. each enum value is initialised only once. This means that you can use == as well as *equals()* to compare enums.

- Enums implicitly extend *java.lang.Enum* and as Java does not support multiple inheritance, an enum cannot extend anything else.

- An enum can also contain fields, methods and constructors. These are called “complex enums”.



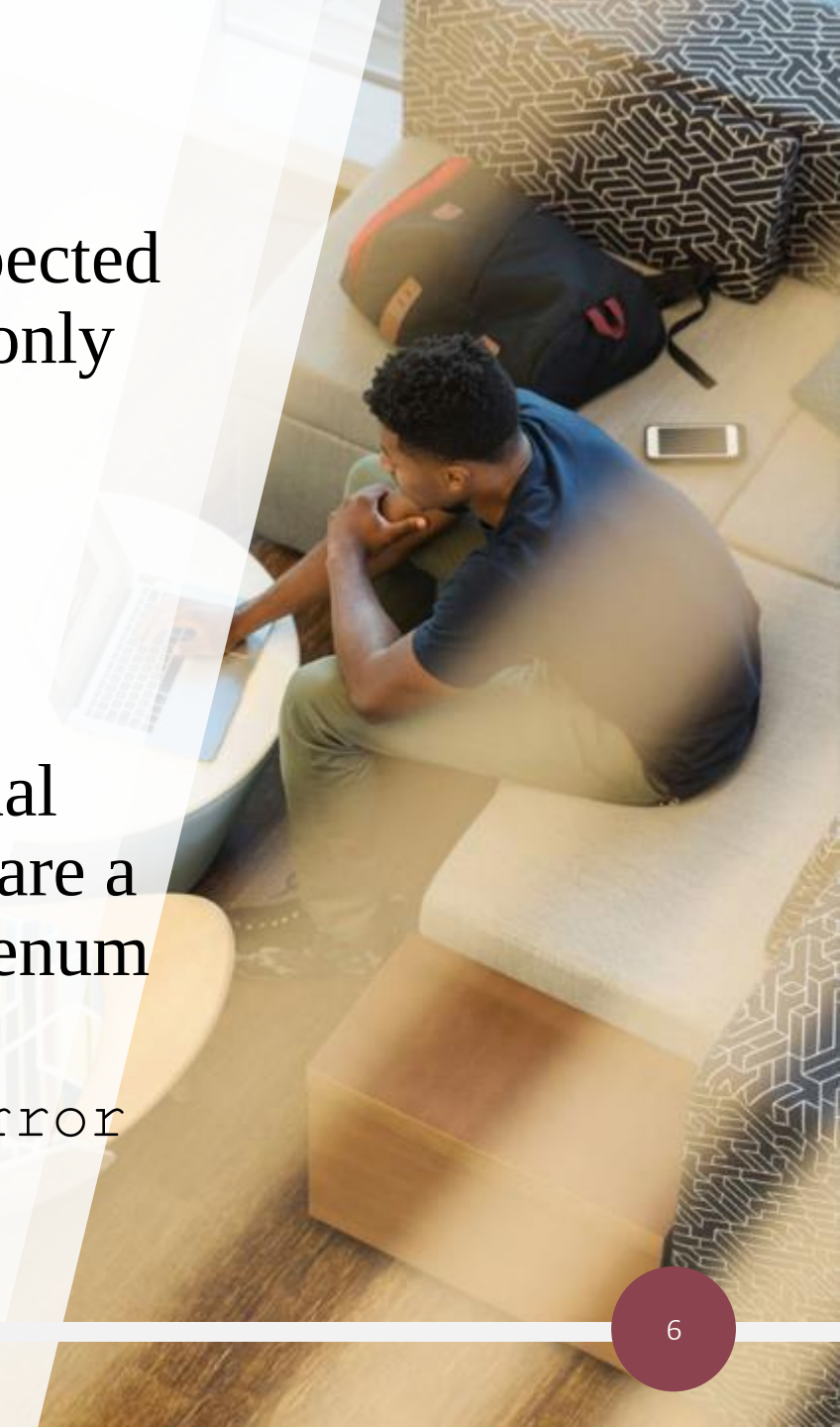
# Enumerations

- Enum constructors are implicitly *private*. This is expected as you cannot extend an enum and constructors can only be called from with the enum (the enum values).

- You cannot *new* an enum.

- Each of the enum values (NORTH etc..) get an ordinal value, starting at 0. However, remember that enums are a type and not a primitive i.e. you cannot compare an enum with an *int*

```
if (Direction.NORTH == 0) // compiler error
```



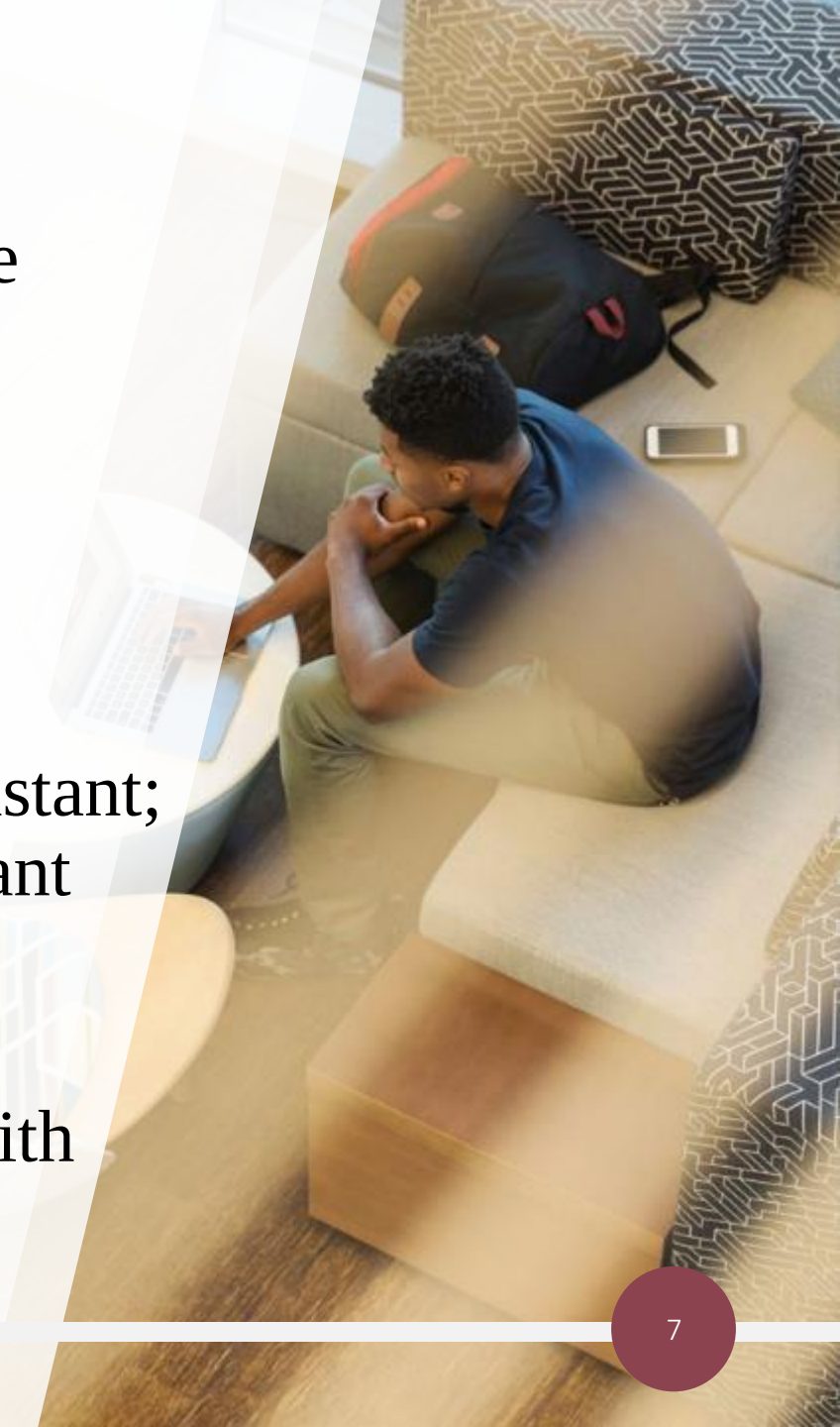
# Enum methods

- *values()* - static *values()* method allows you to iterate through the values of an *enum* in the order they are declared.

- *name()* – returns the name of the enum constant.

- *ordinal()* – returns the ordinal value of the enum constant; its position in the enum declaration; the initial constant has a value of 0.

- *valueOf(String name)* – returns the enum constant with the specified case sensitive name.



# Complex enums

- As stated earlier, enums can have members, constructors and methods.

- We never call the constructors ourselves – we simply refer to the enum value. The first time we refer to an enum value, Java will construct all the enum values. Any reference after that to an enum value, Java simply returns the already constructed one. Therefore, construction only happens once.

- The “constant specific class body” is used whenever you want a particular constant to override a method in the *enum*