Programming Methodology

Lecture 06 –patterns and types

UMASSCS

SCHOOL OF COMPUTER SCIENCE

Objectives

- To learn a little "design pattern" in Java
  - What is a design pattern?
  - The **Singleton** pattern in Java
  - The **Factory Method** in Java
  - How does Scala handle this?
- To learn how to create Scala applications
- To learn about basic types

What is a “Design Pattern”?

In software engineering, a **design pattern** is a general reusable solution to a commonly occurring problem within a given context in software design.

- **Recognizing the problem**
  - Recognizing the solution
- **Different languages offer different solutions**
  - Programming languages are tools
  - Designed for different purposes
  - Design patterns often occur when a language does not provide support to solve a typical problem!

Singletons

In class-based programming, the **singleton** pattern is implemented by creating a class with a method that creates a new instance of the class if one does not exist. If an instance already exists, it simply returns the a reference to that object. To make sure that the object can't be instantiated in any other way, the constructor is made private.

- How do we typically create objects in Java?
  - Yes, we use the **new** operator
- What if we wanted to ensure that only **one** instance of a class exists?
- Why would we want to do this?
- Let us look at an example:
  - **Example:** java-singleton

Singleton in Scala?

- Singleton in Scala?
  - Scala provides language support to handle this!
- Scala has class definitions
  - You define them with **class**
- Scala has object definitions
  - These act as a single object
  - You define them with **object**
- Let us look at an example:
  - **Example:** scala-singleton
  - This example also shows you how to create Scala applications!

Scala Companion Objects

- Scala objects can be associated with a class
  - They provide additional support to a class of the same name.
  - They are typically defined in the same file as the class.
- Let us look at an example:
  - **Example:** scala-checksum-app

Factory Method

In class-based programming, the **factory method** pattern is a *creational pattern* which uses factory methods to deal with the problem of creating objects without specifying the exact class of object that will be created.

- How do we typically create objects in Java?
  - Yes, we use the **new** operator
- We already saw this in the homework!
  - src/main/java/cs220/util/Util.java
- Let us look at another example
  - **Example:** java-singleton-factory-method

Factory Method in Scala?

- Factory method in Scala?
  - Scala provides language support to handle this!
- Scala has class definitions
  - You define them with **class**
- Scala has object definitions
  - These act as a single object
  - You define them with **object**
- Scala has traits
  - These are similar to Java interfaces, but provide much more!
- **Example:** scala-singleton-factory-method

Scala: Basic Types

Value type	Range
`Byte`	8-bit signed two's complement integer (-2^7 to $2^7 - 1$, inclusive)
`Short`	16-bit signed two's complement integer (-2^{15} to $2^{15} - 1$, inclusive)
`Int`	32-bit signed two's complement integer (-2^{31} to $2^{31} - 1$, inclusive)
`Long`	64-bit signed two's complement integer (-2^{63} to $2^{63} - 1$, inclusive)
`Char`	16-bit unsigned Unicode character (0 to $2^{16} - 1$, inclusive)
`String`	a sequence of `Char`s
`Float`	32-bit IEEE 754 single-precision float
`Double`	64-bit IEEE 754 double-precision float
`Boolean`	`true` or `false`

Scala: Literals

- We have seen most of these already!
  - Int: 2, -3, 0xdeadbeef, …
  - Float/Double: 1.234, 1.23e1, …
  - Char: ‘A’, ‘b’, ‘\n’, ‘\101’, ‘\u0041’, …
  - String: “hello”, “””multiline string”””
  - Boolean: true, false

Scala: Symbols

- This one we haven't seen!
  - Symbol: 'Apple
- A symbol has a type:
  - Symbol
- Symbols are *interned*
  - The same symbol it refers to the same object
  - This demonstrates the *flyweight pattern*
    - We will see this later in the course!