Introducing Scala

WCC Knowledge Sharing Session

GR/BLE

Today's Menu

- Scala Background
- Why Scala?
- Type Inferencing/Type System
- Case Classes
- Functional Programming
- Avoiding Null (Billion-Dollar Mistake)
- Pattern Matching
- Live Coding 🎉

Scala Background

- **Sca**lable **La**nguage
- Designed by Martin Odersky
- First release in 2004
- Commercial support by Lightbend

“Scala is a **modern multi-paradigm** programming language designed to express **common** programming **patterns** in a **concise**, **elegant**, and **type-safe** way.”

Why Scala?

- Full interoperability with Java
- Less boilerplate
- Multi-paradigm: functional & object-oriented
- Move away from null
- Multi-core programming

“If I were to pick a language to use today other than Java, it would be Scala.”

– James Gosling (Father of Java)

“I can honestly say if someone had shown me the Programming Scala book by Martin Odersky, Lex Spoon & Bill Venners back in 2003, I'd probably have never created Groovy.”

– James Strachan (creator of Groovy)

Used by

LinkedIn

BBC

theguardian

FOURSQUARE

YAHOO!

Today's Menu

- Scala Background
- Why Scala?
- **Type Inferencing/Type System**
- Case Classes
- Functional Programming
- Avoiding Null (Billion-Dollar Mistake)
- Pattern Matching
- Live Coding 🎉

Type System

Type Inferencing

```
val foo = "Bar"

val answer = 42

val price = 9.99

val nums = List(1, 2, 3)

val map =  Map("John" -> 43, "Jane" -> 36)
```

Type System

Explicit Typing

```
val foo: String = "Bar"

val answer: Int = 42

val price: Double = 9.99

val nums: List[Int] = List(1, 2, 3)

val map: Map[String, Int] =  Map("John" -> 43, "Jane" -> 36)
```

Type System

Variables & Values

```
// variable
var foo = “foo"
foo = “bar” // okay

// value
val bar = “bar”
bar = “foo” // nope
```

Today's Menu

- Scala Background
- Why Scala?
- Type Inferencing/Type System
- **Case Classes**
- Functional Programming
- Avoiding Null (Billion-Dollar Mistake)
- Pattern Matching
- Live Coding 🎉

Case Classes

Java POJO

```
public class Person {
     private String name;
     private LocalDate birthday;

     Person(String name, LocalDate birthday) {
          this.name = name;
          this.birthday = birthday;
     }

     public String getName() {
          return name;
     }

     public LocalDate getBirthday() {
          return birthday;
     }
}
```

Case Classes

Java POJO

```
public class Person {
      private String name;
      private LocalDate birthday;

      Person(String name, LocalDate birthday) {
            this.name = name;
            this.birthday = birthday;
      }

      public String getName() {
            return name;
      }

      public LocalDate getBirthday() {
            return birthday;
      }

      @Override
      public String toString() {
        return name + ", " + birthday;
      }
}
```

Case Classes

Java POJO

```
public class Person {
        private String name;
        private LocalDate birthday;

        Person(String name, LocalDate birthday) {
                this.name = name;
                this.birthday = birthday;
        }

        public String getName() {
                return name;
        }

        public LocalDate getBirthday() {
                return birthday;
        }

        public void setName(String name) {
                this.name = name;
        }

        public void setBirthday(LocalDate birthday) {
                this.birthday = birthday;
        }

        @Override
        public String toString() {
          return name + ", " + birthday;
        }
}
```

Case Classes

Java POJO

```
public class Person {
        private String name;
        private LocalDate birthday;

        Person(String name, LocalDate birthday) {
                this.name = name;
                this.birthday = birthday;
        }

        public String getName() {
                return name;
        }

        public LocalDate getBirthday() {
                return birthday;
        }

        public void setName(String name) {
                this.name = name;
        }

        public void setBirthday(LocalDate birthday) {
                this.birthday = birthday;
        }

        @Override
        public String toString() {
          return name + ", " + birthday;
        }

        @Override
        public boolean equals(Person b) {
                if (b.name.equals(this.name) && b.birthday.equals(this.birthday)) {
                        return true;
                } else {
                        return false;
                }
        }

        @Override
        public int hashCode() {
                return this.name.hashCode() + this.birthday.hashCode();
        }
}
```

Case Classes

Java POJO

```java
public class Person {
	private String name;
	private LocalDate birthday;

	Person(String name, LocalDate birthday) {
		this.name = name;
		this.birthday = birthday;
	}

	public String getName() {
		return name;
	}

	public LocalDate getBirthday() {
		return birthday;
	}

	public void setName(String name) {
		this.name = name;
	}

	public void setBirthday(LocalDate birthday) {
		this.birthday = birthday;
	}

	@Override
	public String toString() {
	  return name + ", " + birthday;
	}

	@Override
	public boolean equals(Person b) {
		if (b.name.equals(this.name) && b.birthday.equals(this.birthday)) {
			return true;
		} else {
			return false;
		}
	}

	@Override
	public int hashCode() {
		return this.name.hashCode() + this.birthday.hashCode();
	}
}
```

Scala Case Class

```scala
case class Person(
  var name: String,
  var birthday: LocalDate
)
```

Case Classes

Java POJO

```java
public class Person {
        private String name;
        private LocalDate birthday;

        Person(String name, LocalDate birthday) {
                this.name = name;
                this.birthday = birthday;
        }

        public String getName() {
                return name;
        }

        public LocalDate getBirthday() {
                return birthday;
        }

        @Override
        public String toString() {
          return name + ", " + birthday;
        }

        @Override
        public boolean equals(Person b) {
                if (b.name.equals(this.name) && b.birthday.equals(this.birthday)) {
                        return true;
                } else {
                        return false;
                }
        }

        @Override
        public int hashCode() {
                return this.name.hashCode() + this.birthday.hashCode();
        }
}
```

Scala Case Class

```scala
case class Person(
  name: String,
  birthday: LocalDate
)
```

Today's Menu

- Scala Background
- Why Scala?
- Type Inferencing/Type System
- Case Classes
- **Functional Programming**
- Avoiding Null (Billion-Dollar Mistake)
- Pattern Matching
- Live Coding 🎉

Functional Programming

- Functions are first-class citizen
- Passing functions as parameters: higher-order functions
- Higher level of abstraction
- It’s awesome

Functional Programming

Java: check if String has uppercase character

```
boolean hasUpperCase = false;

for (int i=0; i < name.length; i++) {
  if (Character.isUppercase(name.charAt(i))) {
    hasUpperCase = true;
    break;
  }
}
return hasUpperCase;
```

Functional Programming

Java: check if String has uppercase character

```
boolean hasUpperCase = false;

for (int i=0; i < name.length; i++) {
  if
  (Character.isUppercase(name.charAt(i)
  )) {
    hasUpperCase = true;
    break;
  }
}
return hasUpperCase;
```

Scala

```
// long version
name.exists{ c: Char =>
  c.isUpperCase
}
```

Functional Programming

Java: check if String has uppercase character

```
boolean hasUpperCase = false;

for (int i=0; i < name.length; i++) {
  if
  (Character.isUppercase(name.charAt(i)
  )) {
    hasUpperCase = true;
    break;
  }
}
return hasUpperCase;
```

Scala

```
// long version
name.exists{ c: Char =>
  c.isUpperCase
}

// short version
name.exists(_.isUpperCase)
```

Functional Programming

Java: sort users by age

```
users.sort(new Comparator {
  @Override
  public int compare(Object user1, Object user2) {
    int userAge1 = ((User) user1).getAge();
    int userAge2 = ((User) user2).getAge();

    if (userAge1 > userAge2) {
      return 1;
    } else if (userAge1 < userAge2) {
      return -1;
    } else {
      return 0;
    }
  }
});
```

Functional Programming

Java: sort users by age

```
users.sort(new Comparator<User> {
  @Override
  public int compare(User user1, User user2) {
    int userAge1 = user1.getAge();
    int userAge2 = user2.getAge();

    if (userAge1 > userAge2) {
      return 1;
    } else if (userAge1 < userAge2) {
      return -1;
    } else {
      return 0;
    }
  }
});
```

Functional Programming

Java: sort users by age

```
users.sort((user1, user2) -> {
  int userAge1 = user1.getAge();
  int userAge2 = user2.getAge();

  if (userAge1 > userAge2) {
    return 1;
  } else if (userAge1 < userAge2) {
    return -1;
  } else {
    return 0;
  }
});
```

Functional Programming

Java: sort users by age

```
users.sort((user1, user2) -> {
  int userAge1 = user1.getAge();
  int userAge2 = user2.getAge();

  if (userAge1 > userAge2) {
    return 1;
  } else if (userAge1 < userAge2) {
    return -1;
  } else {
    return 0;
  }
});
```

Scala

```
def byAge(user1: User, user2: User) =
    user1.age > user2.age

users.sortWith(byAge)
```

Today's Menu

- Scala Background
- Why Scala?
- Type Inferencing/Type System
- Case Classes
- Functional Programming
- **Avoiding Null (Billion-Dollar Mistake)**
- Pattern Matching
- Live Coding 🎉

Avoiding Null

Scala Options

```
val maybeUser: Option[User] = User.findByName("Bob")
// returns Some[User] or None

// if there is no "Bob"
maybeUser == None // returns true

maybeUser.foreach { user =>
  println(user.name)
} // prints "Bob" if he exists, otherwise prints nothing

maybeUser.map(_.name)
         .getOrElse("User not found")
```

Today's Menu

- Scala Background
- Why Scala?
- Type Inferencing/Type System
- Case Classes
- Functional Programming
- Avoiding Null (Billion-Dollar Mistake)
- **Pattern Matching**
- Live Coding 🎉

Pattern Matching

Switch on steroids

```
val maybeUser: Option[User] = User.findByName(“Bob”)

val name = maybeUser match {
     case Some(user) => user.name
     case None => "User not found"
}
```

Pattern Matching

Switch on steroids

```
val maybeUser: Option[User] = User.findByName(“Bob”)

val name = maybeUser match {
     case Some(user) => user.name
     case None => "User not found"
}

def what(thing: Any) = thing match {
  case i: Int => “It’s an Int”
  case s: String => “It’s a String”
  case _ => “I have no clue what it is”
}

what(123) // “It’s an Int”
what(“hello”) // “It’s a String”
what(false) // “I have no clue what it is”
```

Live Coding 🎉

GR/BLE

Thank you!

Questions?

https://github.com/Grible/scala-intro/tree/wcc

steven@grible.co // lennart@grible.co

http://grible.co