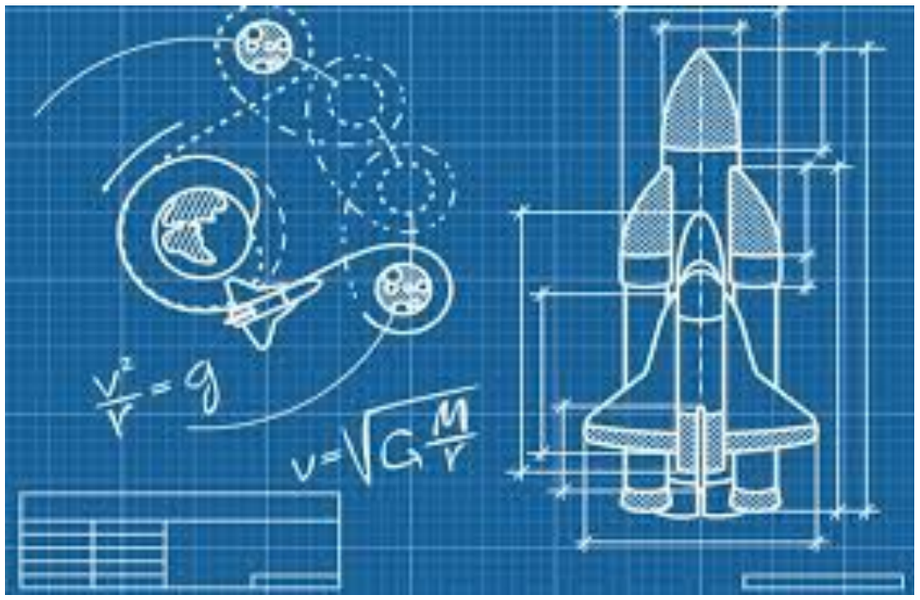
Objects and Data

About Object Oriented Programming in Scala



Class

- Classes in Scala are blueprints for creating objects
- They can contain methods, values, variables, types, objects, traits, and classes which are collectively called members
- Use the class identifier
    - Class names should be capitalized

```
class User

val user1 = new User
```

Example

- Rational numbers, ie a/b
- A new type - Rational
- A constructor, to create elements of this type

```
scala> class Rational(x: Int, y:Int){
     | def numerator = x
     | def denominator = y
     | }
defined class Rational
```

Objects

- Elements of a class type are objects
- Objects are instantiated from a class by a constructor
- You call the constructor, by prefixing the class constructor with operator **new**

```
scala> new Rational(3,4)
res15: Rational = Rational@5c97a373
```

Objects have members

- Selected with “.” infix
- Members have visibility (public and private)

```
scala> new Rational(3,4)
res15: Rational = Rational@5c97a373

scala> res15.numerator
res16: Int = 3
```

Multiple Constructors

- Implicit primary constructor defined by executing the class body

- Can define additional constructors (named this)

```
def this(x:Int) = this(x,1)

val integer = new Rational(1)
//> integer  : objects_and_data.Rational = 1/1
```

Classes have methods

- A method is a function, defined in a class, that can be applied to object instances

```
class Point(var x: Int, var y: Int) {

  def move(dx: Int, dy: Int): Unit = {
    x = x + dx
    y = y + dy
  }
  override def toString: String =
    s"($x, $y)"
}

val point1 = new Point(2, 3)
point1.x  // 2
println(point1)
```

Member Visibility

- public - member can be accessed from outside the class
- private - member can only be accessed from inside the class
- Members are public by default
    - Use the **private** access modifier to hide them from outside of the class

```
private val bound = 100
```

Class parameters

- Classes are parametrized by constructor
- Primary constructor parameters with val and var are public.
- Parameters without val or var are private values, visible only within the class.

```
class Point(x: Int, y: Int)
val point = new Point(1, 2)
point.x  // <-- does not compile
```

Self reference

- What if you need a reference to your object?
- Use the keyword **this**
  - for members, this.x and x are equivalent

```
scala>
```

Preconditions

- Used to enforce a predefined condition on the caller of a function
- Arguments - test and optional message
- Check arguments in constructor
- Impose conditions
  - Fail early - will throw IllegalArgumentException, if test fails

```
require( y!= 0, "Only non zero values for denominator accepted")
```

Assertions

- Similar to require
- **assert**, predefined method
- Takes a test, and optional message
- Throws AssertionError if test fails
    - Different intention than require - used to verify the function itself

```
scala> assert(3>4)
java.lang.AssertionError: assertion failed
  at scala.Predef$.assert(Predef.scala:204)
  ... 28 elided
```

Data Abstraction

- Ability to choose different implementation without affecting clients

- Achieved by encapsulation

High cohesion
Low coupling

Object Oriented Design Rule

Infix operator

- **Infix** notation (math) is characterized by the placement of operators between operands, ie **2+3**

- In Scala, the infix operator is another way of denoting a function call
    - Any method with a single parameter can be used as an infix operator. For example, + can be called with dot-notation

```
// dot notation
10.+(1)
// infix
10 + 1
```

Operators

- In Scala, operators are methods.
- You can use any legal identifier as an operator. This includes a name like **add** or a symbol(s) like **+**
  - Or crazy sequences like !€%&€€ … hmmm … maybe not?

```
class Vec(val x: Double, val y: Double) {
  def +(that: Vec) = new Vec(this.x + that.x, this.y + that.y)
}

val vector1 = Vec(1.0, 1.0)
val vector2 = Vec(2.0, 2.0)

val vector3 = vector1 + vector2
vector3.x  // 3.0
vector3.y  // 3.0
```

Relaxed naming

- alphanumeric, _
- symbolic
- Example a +b

Operator precedence

- Which comes first + or * ?
- Operator precendce - look at first letter, and use the (Java) rules:
  - * / %
  - + -
  - :
  - = !
  - < >
  - &
  - ^
  - |
  - all letters

```
a + b ^? c ?^ d less a ==> b | c
```

type alias

In Scala you can create a simple alias for a more complex type.

```
type Row = List[Int]
type Matrix = List[Row]
// instead of List[List[Int]]
```

Lab

objects_and_data_05