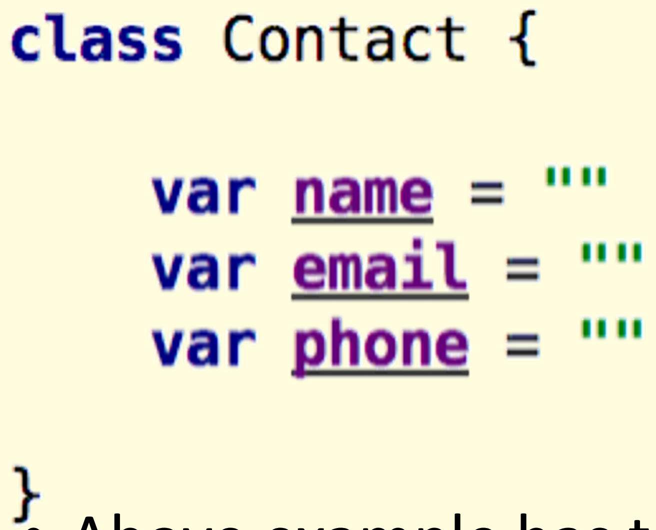
Kotlin classes

CO5225 – Andrew Muncey

Contents

Basic class syntax



```
class Contact {

    var name = ""
    var email = ""
    var phone = ""

}
```

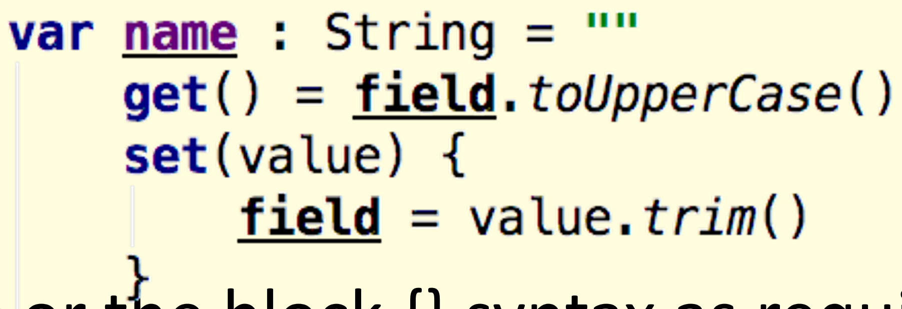
- Above example has three properties
- Fields (ivars) created as above but with `private` access modifier
- All fields / properties of a class must be instantiated
  - Either at declaration, in the constructor, or in an init block
- Classes and the methods within are public by default

Getters and Setters

- In Java, we were forced to think about whether properties had both a getter and setter
  - Backing fields are not generated by default in Kotlin
- In Kotlin, if we do nothing, they will have both
- Quite often we want the ‘set’ to be private
  - Use `private set` declaration
- Don’t forget that if the value is immutable (i.e. set at object initialization and never changed), we can use `val`

Custom getters and setters

- A backing field will be created if required by a custom getter or setter (as we would create manually in Java)
- E.g.

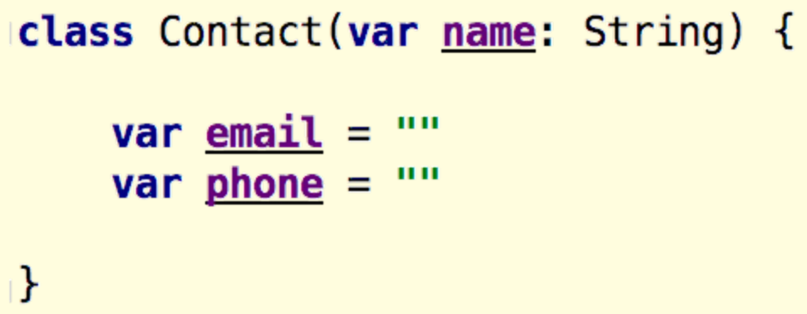


```
var name : String = ""
    get() = field.toUpperCase()
    set(value) {
        field = value.trim()
    }
```

- Use = or the block {} syntax as required
- *Must* use `field` to access backing field
- *Convention* is to use `value` in setter
- Must initialise field inline (cannot use constructor / init block)

Primary Constructor

- Can declare and initialise properties
  - May result in properties being declared in two places
- Cannot perform operations
  - Possible to use `init` block in some cases, but not always possible or best practice
    - Immutable properties whose values would need to be determined by some operation could not go in primary constructor if initialised differently in a secondary constructor
- Primary constructor is not a requirement
- Demo – primary constructor



```
class Contact(var name: String) {

    var email = ""
    var phone = ""

}
```

`init` block

- Additional setup which executes after the constructor
- Code placed inside braces following `init` keyword
- Multiple `init` blocks permitted
  - Execute in the order they appear in the code

Secondary constructors

- Declared using `constructor` keyword

```
constructor(name: String, email: String, phone: String){
    this.name = name
    this.email = email
    this.phone = phone
}
```

- If a primary constructor exists, each secondary constructor must call it, either directly or indirectly

```
constructor(name: String, email: String, phone: String) : this(name){
    this.email = email
    this.phone = phone
}
```

Demo

- Conversion of Employee class (next slide, from Java Classes lecture)
- Conversion of PokeTutor class (from Java tutorial) to Kotlin

```java
package uk.ac.chester;
import java.util.Date;

class Employee {

    private String forename, surname;
    private boolean current;
    private Date startDate, endDate;

    //region Constructors
    Employee(String name, String surname){

        this.forename = name;
        this.surname = surname;
        current = true;
        startDate = new Date();
    }
    //endregion

    //region Accessors and Mutators
    public String getForename() { return forename; }

    public void setForename(String name) { forename = name.trim(); }

    public String getSurname() { return surname; }

    public void setSurname(String surname) { this.surname = surname; }

    public boolean isCurrent() { return current; }

    public Date getStartDate() { return startDate; }

    public Date getEndDate() {
        //may error - revisit to handle later
        return endDate;
    }
    //endregion

    //region Methods
    public String getFullName() {
        return forename + " " + surname;
    }

    public void fire(){
        current = false;
        endDate = new Date();
    }
    //endregion

}
```

```java
package uk.ac.chester;
import java.util.Date;

class Employee {

    private String forename, surname;
    private boolean current;
    private Date startDate, endDate;

    //region Constructors
    Employee(String name, String surname){

        this.forename = name;
        this.surname = surname;
        current = true;
        startDate = new Date();
    }
    //endregion

    //region Accessors and Mutators
    public String getForename() { return forename; }

    public void setForename(String name) { forename = name.trim(); }

    public String getSurname() { return surname; }

    public void setSurname(String surname) { this.surname = surname; }

    public boolean isCurrent() { return current; }

    public Date getStartDate() { return startDate; }

    public Date getEndDate() {
        //may error - revisit to handle later
        return endDate;
    }
    //endregion

    //region Methods
    public String getFullName() {
        return forename + " " + surname;
    }

    public void fire(){
        current = false;
        endDate = new Date();
    }
    //endregion

}
```

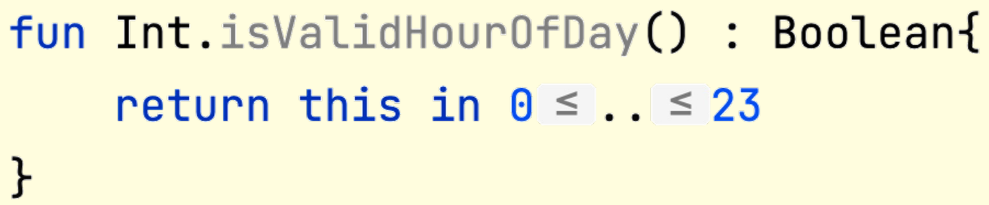
```kotlin
import java.util.Date

class Employee (var forename: String, var surname: String) {

    val startDate : Date

    var endDate : Date?
        private set

    var current : Boolean
        private set

    init{

        current = true
        startDate = Date()
        endDate = null
    }

    fun fullName() : String{
        return "$forename $surname"
    }

    fun fire() {
        current = false
        endDate = Date()
    }

}
```

Data classes

- Uses properties defined in a primary constructor to automatically provide implementations of:
  - `equals()` & `hashCode()`
  - `toString()`
  - `copy()`
- Prefix the class declaration with the `data` keyword
- Ensure primary constructor includes at least one property and that all parameters are properties
  - Above functions generated based on primary constructor properties alone
- Cannot be open

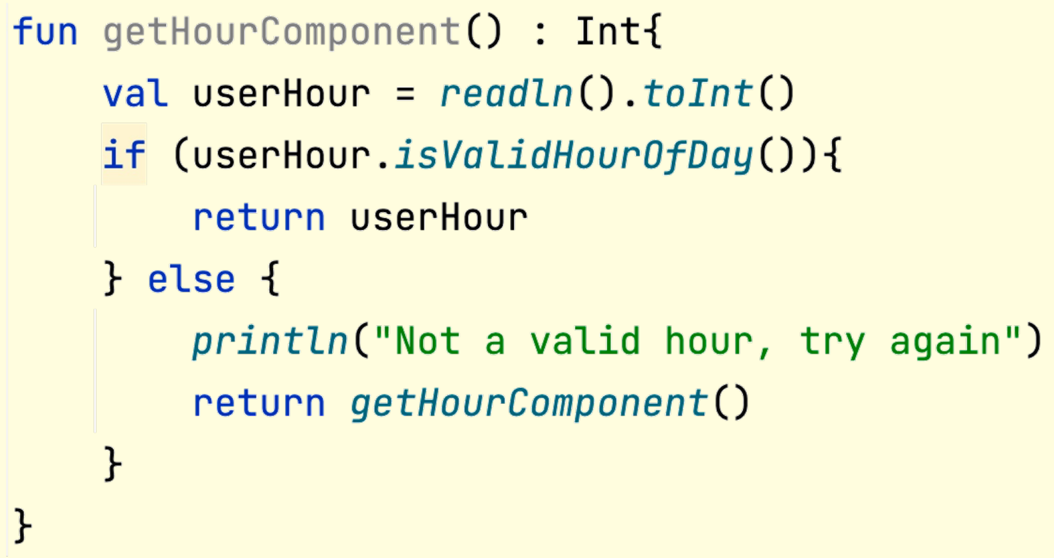
Extension Functions

- Allows us to add functions an existing class, without extending it
- Prepend the class name to the method, with a dot



```
fun Int.isValidHourOfDay() : Boolean{
    return this in 0 ≤ .. ≤ 23
}
```

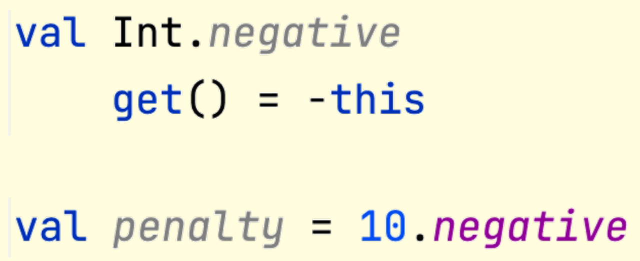
- Usage:



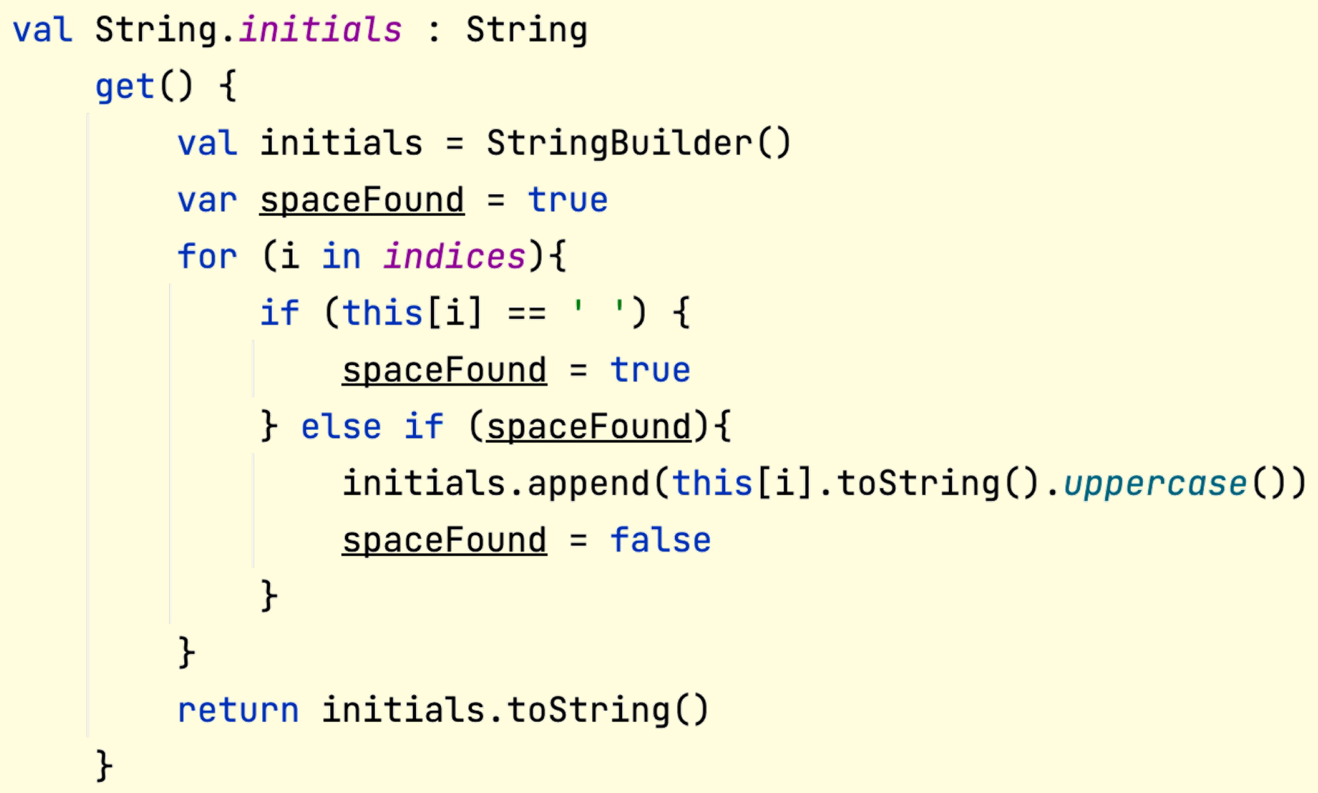
```
fun getHourComponent() : Int{
    val userHour = readln().toInt()
    if (userHour.isValidHourOfDay()){
        return userHour
    } else {
        println("Not a valid hour, try again")
        return getHourComponent()
    }
}
```

Extension Properties

- Cannot add backing fields so typical use is for calculated values



```kotlin
val Int.negative
    get() = -this

val penalty = 10.negative
```



```kotlin
val String.initials : String
    get() {
        val initials = StringBuilder()
        var spaceFound = true
        for (i in indices){
            if (this[i] == ' ') {
                spaceFound = true
            } else if (spaceFound){
                initials.append(this[i].toString().uppercase())
                spaceFound = false
            }
        }
        return initials.toString()
    }
```