# Sales And Care Tool Box(SCT)

Care Toolbox enables actions and processes that manage and support customer information. Toll Box to allow Customer Service Representatives (CSRs) a full 360º view of the customer. The toolbox allows CSRs to view and update customer information such as customer details, products and services, order history and financial information.

Following list of modules are there in SCT

1. Overview
2. Customer Information
3. Agreements
4. Financial
5. Cases
6. Shops
7. Orders
8. History
9. **Overview**

*Overview* gives you a quick survey of the customer and acts as a starting point to view more specific details.

* Subscription details
* Balance details
* Cases
* Resource
* Activities
* Orders

1. **Customer Information**

The *Customer info* view includes customer-related information

* Basic details
* Contact details
* Marketing

1. **Agreements**

The Agreements shows the list of all agreement customer having

* Agreements list
* Agreement details
* Agreement items
* Additional tools

1. **Financials**

The *Financials* view includes all customer-related information such as customer's billing accounts and other billing-related information

* Transaction history
* Billing accounts

1. **Cases**

In the *Cases* view you can view customer's case information: a list of the cases related to the customer, and the details of the case that is currently selected.

You can filter the cases by using the **Status** filter drop-down menu. The following status filters are available:

* All
* Active
* Pending
* Terminated
* In Progress

1. **Shops**

In the *Shop* view you can log in and transfer to Sales Toolbox while maintaining the customer information selected in the Care Toolbox

1. **Orders**

In the *Orders* view you can view customer's order information and baskets. You can also filter the list of orders using various criteria such as MSISDN and status.

The following information is displayed in the Orders view:

* Order ID
* Details
* Contact details
* Delivery details
* Sales Details
* Price

1. **History**

In the *Event Lo*g view you can view the usage events of the customer. Events in the list can also be filtered using various criteria

Transactional Document Management(TDM)

Transactional Document Management System (TDMS) is solution for telecommunications activities to create, handle and manage documents such as bills, invoices, contract according to Customer Service Provider (CSP) business processes.

Basic functionalities of two TDMS components

* Document Storage
* Document Manager.

**Document Storage :** Stores and archives documents with defined retention policies, imports documents with configurable policies and allows application to retrieve documents

* Upload Documents
* Store uploaded documents to data storage
* Access point for other systems to retrieve documents

In document storage having two modules

1. **Documents :** Here we can see whatever uploaded documents.

* The user can search documents stored in the storage
* Only Active documents will be shown in the initial search results, the user can modify the search results by using the **Filters** button.
* The user can filter by **Status**, **Type** and **Search date range**. The user can also do mass operations, e.g. select multiple documents and **Archive**, **Restore** or **Delete permanently**.
* The user can select a document and see the Details and the Change history.
* The user can Delete the document or Delete permanently from the More drop-down menu
* After the document has been deleted the user can Restore or Delete permanently.

1. **Upload Documents**

* The end user can **Drag and drop a file here** or **Select file** to upload. The Upload button will become active once the document has been selected
* After the document is selected the user can edit the name and can select 1) **Clear fields** 2) **Validate** 3) **Upload**.

**Document Manager :** Performs generating and managing of the document templates that are used in various applications of BSS system; creates formatted documents from templates and  delivers documents in desired format

* Create and edit templates
* Automatic preview
* Store templates to the database

In document manager having 4 modules

1. **Templates**

* Here we can see all templates based on filters
* Create new templates
* We can view template details

1. **Snipper library**

* Here we can see all snipper based on filters
* Create new snipper
* We can view snipper details

1. **Assets**
2. **Admin tools**

* Create/delete document type
* Create/delete customer segment
* Create/delete category
* General settings

Scalable Language

* **Introduction**

[Scala](https://www.geeksforgeeks.org/scala-programming-language/) is a general-purpose, high-level, multi-paradigm programming language. It is a pure object-oriented programming language which also provides the support to the functional programming approach. There is no concept of primitive data as everything is an object in Scala

**Note:**People always thinks that Scala is a extension of [Java](https://www.geeksforgeeks.org/java/). But it is not true. It is just completely interoperable with Java

* Features of Scala
  + Type inference
  + Singleton object
  + Immutability
  + Lazy computation
  + Case classes and pattern matching
  + Concurrency control
  + String interpolation
  + Higher order function
  + Traits
  + Rich collection set
* **Difference between Scala Functions & Methods:** Function is a object which can be stored in a variable. But a method always belongs to a class which has a name, signature bytecode etc. Basically, you can say a method is a function which is a member of some object
* **Pattern matching :**Pattern matching is a way of checking the given sequence of tokens for the presence of the specific pattern. It is the most widely used feature in Scala. It is a technique for checking a value against a pattern. It is similar to the [*switch statement of Java*](https://www.geeksforgeeks.org/switch-statement-in-java/) and [*C*](https://www.geeksforgeeks.org/switch-statement-cc/)
* **Yield Keyword : yield** keyword will returns a result after completing of loop iterations. The [for loop](https://www.geeksforgeeks.org/for-loop-in-scala/) used buffer internally to store iterated result and when finishing all iterations it yields the ultimate result from that buffer
* **Class** : A class is a user-defined blueprint or prototype from which objects are created. Or in other words, a class combines the fields and methods(member function which defines actions) into a single unit
* **Object** : Object is a basic unit of Object Oriented Programming and represents the real-life entities. A typical Scala program creates many objects, which as you know, interact by invoking methods

**Ex** : class Student(id : Int,name : String){

}

Var student=new Student(101,”Krishnam”)

* **Inheritance** : Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in Scala by which one class is allowed to inherit the features(fields and methods) of another class.

**EX :** class Student 1{

Var schoolName : Strnig =”My School”

}

Class Student2 extends Student1 {

}

* **Abstraction : abstraction** is the process to hide the internal details and showing only the functionality. In Scala, abstraction is achieved by using an abstract class

**EX :** abstract class myauthor

{

          def details()

}

 class GFG extends myauthor

{

    def details()

    {

    }

}

* **Singleton Object :** Scala is more object oriented language than Java so, Scala does **not** contain any concept of **static** keyword. Instead of static keyword Scala has **singleton object**. A Singleton object is an object which defines a single object of a class. A singleton object provides an entry point to your program execution. If you do not create a singleton object in your program, then your code compile successfully but does not give output. So you required a singleton object to get the output of your program. A singleton object is created by using **object** keyword

**Ex :** object student{

}

* **Companion Object :** **Companion object** is known as an object whose name is same as the name of the class. Or In other words, when an object and a class have the **same name**, then that object is known as the companion object and the class is known as companion class. A companion object is defined in the same source file in which the class is defined. A companion object can access both private methods and private fields of the class.

**Ex :** class Student{

}

Object Student{

}

* **Constructor :**

1. **Primary Constructor :** When our Scala program contains only one constructor, then that constructor is known as a primary constructor. The primary constructor and the class share the same body, means we need not to create a constructor explicitly.

**EX :** class Student(id : Int, name : String)

1. **Auxiliary Constructor :** In a Scala program, the constructors other than the primary constructor are known as auxiliary constructors. we are allowed to create any number of auxiliary constructors in our program, but a program contains only one primary constructor.

**Ex :**

class GFG( Aname: String, Cname: String)

{

    // Auxiliary Constructor

    def this(Aname: String, Cname: String, no:Int)

    {

        // Invoking primary constructor

        this(Aname, Cname)

        this.no=no

    }

}

* A **Case Class** is just like a regular class, which has a feature for modeling unchangeable data. It is also constructive in pattern matching. It has been defined with a modifier **case**, due to this case keyword.
* **Case Object :** A **Case Object** is also like an object, which has more attributes than a regular Object. It is a blend of both case classes and object. A case object has some more features than a regular object.  
  Below two are important features of case object:
  + It is serializable.
  + It has a by default hashCode implementation.
* **Polymorphism :**  is the ability of any data to be processed in more than one form.
* **Value classes** are a new mechanism which help to avoid allocating run time objects. **AnyVal** define value classes. Value classes are predefined, they coincide to the primitive kind of Java-like languages.  
  There are nine predefined value types : Double, Float, Long, Int, Short, Byte, Char, Unit, and Boolean
* **Multi threading :** A process in which multiple threads executing simultaneously that is called **multithreading**. It allows you to perform multiple tasks independently
* **Currying: Currying** in Scala is simply a technique or a process of transforming a function. This function takes multiple arguments into a function that takes single argument. It is applied widely in multiple functional languages.

**Ex :** def functionName(arg1)=(arg2) => operation

* **anonymous function** is also known as a function literal. A function which does not contain a name is known as an anonymous function. An anonymous function provides a lightweight function definition. It is useful when we want to create an inline function

**(x : Int,y : Int) => x\*y**

**(\_: Int )\*(\_ : Int)**

* **Higher Order Function**  : A function is called **Higher Order Function** if it contains other functions as a parameter or returns a function as an output i.e, the functions that operate with another functions are known as Higher order Functions

**Ex :** val num = List(7, 8, 9)

def multiplyValue = (y: Int) => y \* 3

val result = num.map(y => multiplyValue(y))

* **Nested Function** : we can define functions inside a function and functions defined inside other functions are called **nested or local functions**

**Ex :** def function1(){

def function2(){

}

Function2()

}

* **Partially applied functions**  : The **Partially applied functions** are the functions which are not applied on all the arguments defined by the stated function i.e, while invoking a function, we can supply some of the arguments and the left arguments are supplied when required. we call a function we can pass less arguments in it and when we pass less arguments it does not throw an exception. these arguments which are not passed to function we use hyphen( \_ ) as placeholder.

Ex : def sum(a : Int, b : Int)=a+b

Var s10=sum(10,\_: Int)

S10(20)

* **String Interpolation** refers to substitution of defined variables or expressions in a given String with respected values. String Interpolation provides an easy way to process String literals. To apply this feature of Scala, we must follow few rules:
  + String must be defined with starting character as **s** / **f**/**raw**.
  + Variables in the String must have ‘$’ as prefix.
  + Expressions must be enclosed within curly braces ({, }) and ‘$’ is added as prefix

Ex : val str=s”sum of $x and $y is ${x+y}”

* **Trait** : Traits are like [interfaces in Java](https://www.geeksforgeeks.org/interfaces-in-java/). But they are more powerful than the interface in Java because in the traits you are allowed to implement the members. **Traits** can have methods(both abstract and non-abstract), and fields as its members.
* **Tuple** : is a collection of elements. Tuples are heterogeneous data structures, i.e., is they can store elements of different data types. A tuple is immutable, unlike an array in scala which is mutable.

**Val name=(15,”Mena” ,true)**

* The ***Option*** in Scala is referred to a carrier of single or no element for a stated type. When a method returns a value which can even be null then Option is utilized i.e, the method defined returns an instance of an Option, in place of returning a single object or a null

Play Frame Work

Induction Training

Courses