Created by: Zoltán Richárd Jánki
Date: 14.04.2021

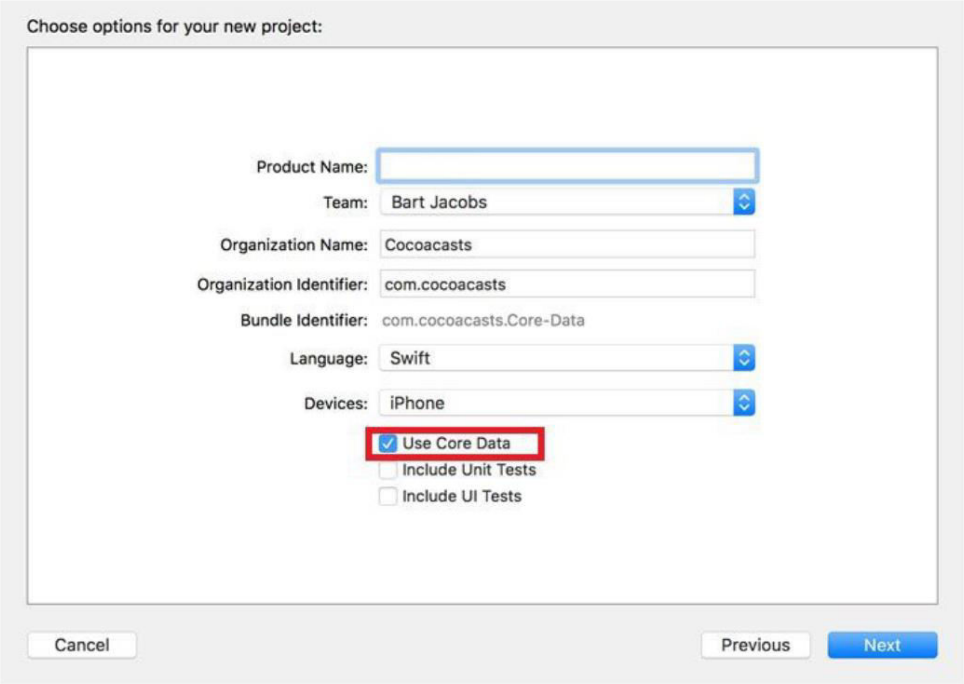
Apple Swift Course
Practice 9

Core Data:

- a framework that manages objects in the model layer of the application
- it gives a general and automatized solution for managing the lifecycle of objects and the object graphs
- persistence is implemented
- no SQL codes are needed

Creating an application with CoreData:

- when creating a new project, put a tick in the box of **"Use Core Data"** option
  ~ we get an extended project (extended *AppDelegate.swift* file and a new *.xcdatamodeld* file)



~ never use CoreData or Core Data as project name because the library that contains the implementation of the CoreData framework uses the same name, and an import statement will be ambiguous.

How to use CoreData:

- CoreData lib must be imported

```
import CoreData
```

- create a data model for creating a database instance
  ~ open the ProjectName.xcdatamodeld file
  ~ create an entity (Add Entity button)
  ~ add the attributes to the Entity (+), set the type of attribute (Integer, String, etc.)
- create an array for the objects in your source code (in order to manage object locally)

```
var items = [NSManagedObject]()
```

  ~ the type is array of NSManagedObjects because the CoreData records use this type

Created by: Zoltán Richárd Jánki
Date: 14.04.2021

- **saving** a record in database:
  ~ access the AppDelegate instance in order to create a context and access the objects and methods of AppDelegate class
```
let appDelegate = UIApplication.shared.delegate as? AppDelegate
```
  ~ create a context – (it works as a sheet of paper that can be used to create and modify objects)
```
let managedContext = appDelegate.persistentContainer.viewContext
```
  ~ access the entity in the database
```
let entity = NSEntityDescription.entity(forEntityName:
"EntityName", in: managedContext)!
```
  ~ create a database record object connected to the entity
```
let record = NSManagedObject(entity: entity, insertInto:
managedContext)
```
  ~ the values of record fields can be modified by using their name (e.g.: if the attribute name is **stringAttr** and its type is **string**)
```
record.setValue("someString", forKeyPath: "stringAttr")
```
  ~ save the context (this is the persistence)
```
do {
  try managedContext.save()
  items.append(record)
} catch let error as Error {
  print(error)
}
```
- **reading** records from database:
  ~ access the AppDelegate instance in order to create a context and access the objects and methods of AppDelegate class
```
let appDelegate = UIApplication.shared.delegate as? AppDelegate
```
  ~ create a context – (it works as a sheet of paper that can be used to create and modify objects)
```
let managedContext = appDelegate.persistentContainer.viewContext
```
  ~ create a Fetch request
```
let fetchRequest = NSFetchRequest<NSManagedObject>(entityName:
"entityName")
```
  ~ do fetching and put the results into our local array (**items**)
```
do {
  items = try managedContext.fetch(fetchRequest)
} catch let error as Error {
  print(error)
}
```

- **deleting** records from database:
  ~ access the AppDelegate instance in order to create a context and access the objects and methods of AppDelegate class
```
let appDelegate = UIApplication.shared.delegate as? AppDelegate
```
  ~ create a context – (it works as a sheet of paper that can be used to create and modify objects)
```
let managedContext = appDelegate.persistentContainer.viewContext
```
  ~ if the concrete NSManagedObject is available that we want to delete, we can call the **delete()** method immediately (e.g.: delete the 6th element from the list ***(indices start from 0!)***)
```
managedContext.delete(items[5])
```
  ~ do not forget to keep the array up-to-date
```
items.remove(at: 5)
```
  ~ save the context (this is the persistence)

```
do {
  try managedContext.save()
} catch let error as Error {
  print(error)
}
```