TODAY

- Core Data
  - maps
  - tableview
CORE DATA

- Object model
  - kind of like an object-relational mapping (ORM) for DBs
- Basic ideas
  - is a database on your device
    - but you just build graph of objects
    - graph is automatically serialized and saved in database
    - you create “fetches” that are turned into DB queries
  - performant
    - very fast
    - very functional (sophisticated queries), mapped to SQLite (most of the time) underneath the hood.

CREATING

- Either create new project with core data selected, or
- Add Cocoa File / Core-Data / Data Model
  - And copy container and saveContext() from AppDelegate

- A “context” is a database
  - normally use just one

- Creates “<name>.xcdatamodeld”
  - view / edit graphically!
THINGS TO COME

CORE DATA

• Entities
  • classes (tables)
• Attributes
  • properties (columns)
• Relationships
  • references (foreign keys)
• Access through an **NSManagedObjectContext**
  • from a **NSPersistentContainer**
    `(UIApplication.shared.delegate as! AppDelegate).persistentContainer`
THAT CONTEXT

• Need an **NSManagedObjectContext** to use **Core Data**
  • Only use this on the main queue
    • **contexts** are **not** thread-safe
    • main queue is single-threaded
    • do not share contexts across queues
    • look in AppDelegate to make additional contexts

```swift
let cntr = (UIApplication.shared.delegate as! AppDelegate).persistentContainer
let context: NSManagedObjectContext = cntr.viewContext
```

STARTING, CONTEXTS

**DEMO**
## INSERTING (OLD WAY)

```swift
let context = AppDelegate.viewContext
let coord: NSManagedObject = NSEntityDescription.insertNewObject(
    forEntityName: "Location",
    into: context)
```

- **Database objects:**
  - all represented by NSManagedObjects or subclasses
  - NSManagedObjects is an Entity
  - Attributes are nil until explicitly instantiated
- **Entity attributes:**
  ```swift
  let name = ride.value(forKeyPath: "distance") as? Double
  ride.setValue("Co-Motion Divide", forKeyPath: "bike")
  ```
- **Ugly!** What if we could just
  ```swift
  ride.distance = 38.4
  _ = ride.bike
  ```

---

## TYPES

- **Numbers**
  - *Double, Int, etc.*
  - if “Use Scalar Type” checked in data model editor in XC
- **Binary data**
  - *NSData’s*
- **Dates**
  - *NSDate’s*
- **Relationships**
  - To-many are *NSSet’s*
  - To-one are *NSManagedObject’s*
CHANGES

• Only in memory until you save:

  • try context.save() // throws

CREATING ENTITIES IN XC

DEMO
HOW TO CREATE AN OBJECT?

```swift
let ride = Ride(context: AppDelegate.viewContext)
ride.name = name
ride.when = Date()
ride.distance = distance

for loc in allPoints {
    let obj = Location(context: AppDelegate.viewContext)
    obj.latitude = loc.coordinate.latitude
    obj.longitude = loc.coordinate.longitude
    obj.when = loc.timestamp
    obj.ridename = name
    obj.ride = ride  // <--- RELATIONSHIP,
                   // or "ride.addToLocations(obj)"
                   // same thing
}

do {
    try AppDelegate.viewContext.save()
} catch {
    print("loc save fail")
}
```

• type-checking!

MORE CONTEXTS

• Every object know’s it’s context:

```swift
let ride = Ride(context: location.managedObjectContext)
```

• Dot notation extends to relationships:

```swift
let theRideName = location.ride.name
```
DELETION

- `managedObjectContext.delete(_: object: location)`

```swift
// can define a function in entity class
func prepareForDeletion() {
    // update counts of instances, for example
}
```

- Deletion semantics
  - null-out relationship
  - cascading deletes

QUERIES

- How to make a query:
  - `create the request request: NSFetchRequest`
    ```swift
    let fetchRides: NSFetchRequest<Ride> = Ride.fetchRequest()
    ```
  - set of **Entities** to fetch (single type)
  - `NSSortDescriptors` for ordering of results
    ```swift
    set of descriptors [first-sort, second-sort]
    ```
  - `NSPredicate` boolean for filtering
QUERYING: SORT DESCRIPTORS

• How to make a sort descriptor:

```swift
let sortDescriptor = NSSortDescriptor(
    key: "ride-name", ascending: true,
    selector: #selector(NSString.localizedStandardCompare(_:)) // default
)
```

• Considerations
  • many pre-defined compare funcs (ignore case, localized, etc.)
  • predefined compare funcs executed in database (fast)
  • you can pass your own Swift func (slow)
    • because all data must be copied from DB to compare

QUERYING: NSPREDICATE

• How to make a predicate
  • use `String(format: ...)` syntax

```swift
let predicate = NSPredicate(format: "name = %@ && registered > %",
                              nm, when)
```

• Considerations
  • Format-like syntax similar to “bind variables” in db-land
    • strong separation of values from predicate structure
    • defeats most SQL injection attacks
EXECUTING A QUERY

- Execute the query:
  
  ```swift
  let resr = try? AppDelegate.viewContext.fetch(fetchRides)
  ```
  
- Throws
- Returns (possibly empty) array
- Faults data in:
  - data not necessarily copied from DB until attributes accessed

BACK TO BIKE TRAVELS

DEMO