Lecture 7

Modeling & Design Review

Class Diagram Review

UI Design Review

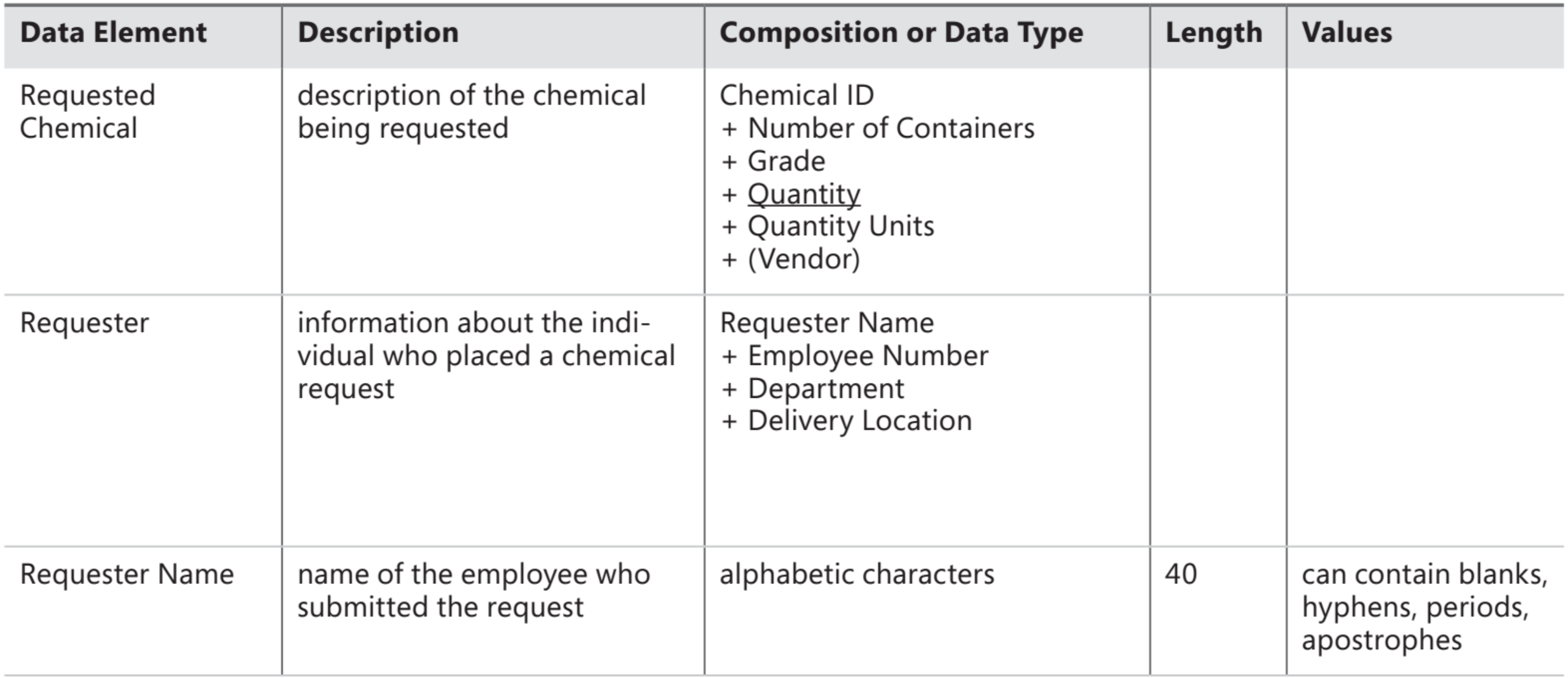
Topics covered

- ✧ Modeling & Design Review
  - Data Dictionary
  - Class Diagram
  - Mockup & Prototype
- ✧ JPA Query Methods
- ✧ JPA Criteria API

Data Dictionary

Data Dictionary

- An E-R model is usually insufficient for describing details, special cases, and domain concepts.
- The *data dictionary* represents data elements and structures of the application domain.



Data Element	Description	Composition or Data Type	Length	Values
Requested Chemical	description of the chemical being requested	Chemical ID + Number of Containers + Grade + Quantity + Quantity Units + (Vendor)		
Requester	information about the individual who placed a chemical request	Requester Name + Employee Number + Department + Delivery Location		
Requester Name	name of the employee who submitted the request	alphabetic characters	40	can contain blanks, hyphens, periods, apostrophes

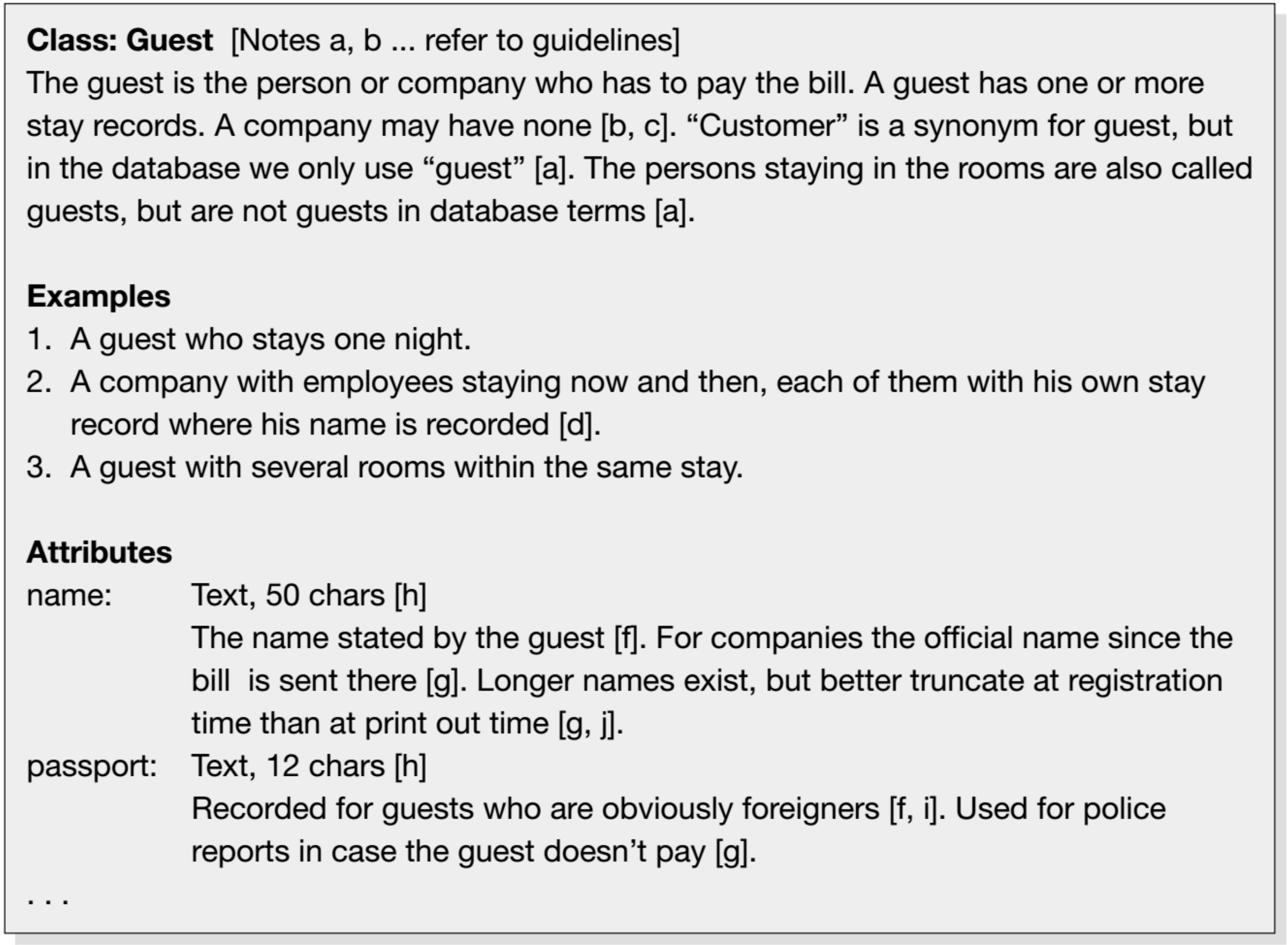
Data Dictionary contains

- Description of data elements
- Data validation criteria
- Composition
- Data types
- Allowed values
- Data examples

Data Dictionary

- Data Dictionary should be structured according to the data model (for ease of reading).
- It should contain non-trivial details.
- **Advantages:**
  - It helps avoid the problem when project participants have different understandings of the data.
  - It can be a good *supplement* to the E-R diagram
  - Requirements validation: customers and expert users can validate the description through the data dictionary.

Data Dictionary example 1



Class: Guest [Notes a, b ... refer to guidelines]
The guest is the person or company who has to pay the bill. A guest has one or more stay records. A company may have none [b, c]. “Customer” is a synonym for guest, but in the database we only use “guest” [a]. The persons staying in the rooms are also called guests, but are not guests in database terms [a].

Examples

1. A guest who stays one night.
2. A company with employees staying now and then, each of them with his own stay record where his name is recorded [d].
3. A guest with several rooms within the same stay.

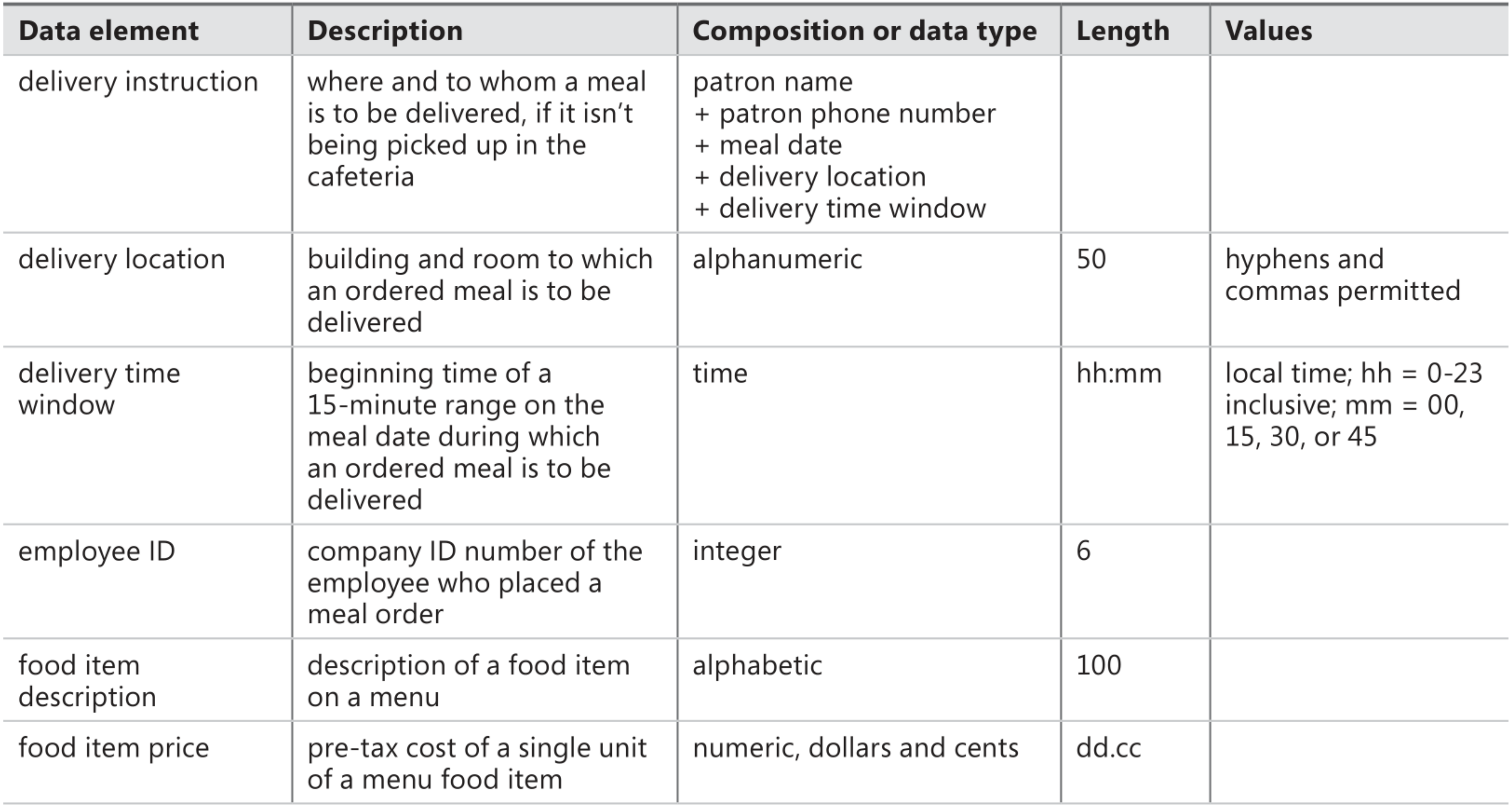
Attributes

name: Text, 50 chars [h]
The name stated by the guest [f]. For companies the official name since the bill is sent there [g]. Longer names exist, but better truncate at registration time than at print out time [g, j].

passport: Text, 12 chars [h]
Recorded for guests who are obviously foreigners [f, i]. Used for police reports in case the guest doesn’t pay [g].

. . .

Data Dictionary example 2



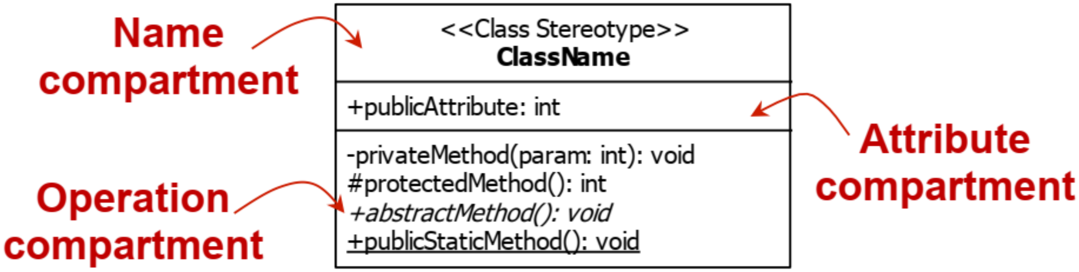
Data element	Description	Composition or data type	Length	Values
delivery instruction	where and to whom a meal is to be delivered, if it isn't being picked up in the cafeteria	patron name + patron phone number + meal date + delivery location + delivery time window		
delivery location	building and room to which an ordered meal is to be delivered	alphanumeric	50	hyphens and commas permitted
delivery time window	beginning time of a 15-minute range on the meal date during which an ordered meal is to be delivered	time	hh:mm	local time; hh = 0-23 inclusive; mm = 00, 15, 30, or 45
employee ID	company ID number of the employee who placed a meal order	integer	6	
food item description	description of a food item on a menu	alphabetic	100	
food item price	pre-tax cost of a single unit of a menu food item	numeric, dollars and cents	dd.cc	

UML Class Diagram Review

Class Diagram Revision

- Class diagrams exist at a lower level of abstraction than component diagrams.
  - ✓ Models consisting of classes and relationships between them necessary to achieve a system's functionality.
  - ✓ Weather a class diagram is created or not, the code of an object-oriented systems will always reflect some class design.
  - ✓ Therefore, there is two-way relationship between class diagrams and object-oriented code.
    - Class diagrams can be transformed to code (i.e., forward engineering)
    - Code can be transformed into class diagrams (i.e., reverse engineering)
  - ✓ This makes class diagrams the most powerful tool for designers to model the characteristics of object-oriented software before the construction phase.

- To become an effective designer, it is essential to understand the direct mapping between class diagrams and code. Let's take a closer look at the fundamental unit of the UML class diagram: the class.



Name compartment

<<Class Stereotype>>
ClassName

+publicAttribute: int

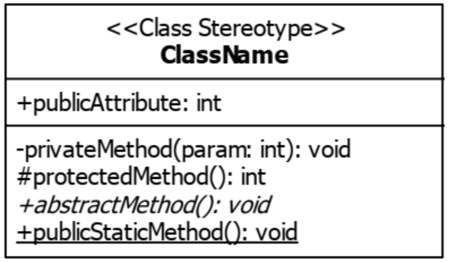
-privateMethod(param: int): void
#protectedMethod(): int
+abstractMethod(): void
+publicStaticMethod(): void

Attribute compartment

Operation compartment

Class Diagram Revision

- Name compartment
  - ✓ Reserved for the class name and its stereotype
  - ✓ Class names can be qualified to show the package that they belong to in the form of Owner::ClassName.
  - ✓ Commonly used stereotypes include:
    - <<interface>>
      - Used to model interfaces.
    - <<utility>>
      - Used to model static classes.
- Attribute compartment
  - ✓ Reserved for the class' attribute specification.
    - Including name, type, visibility, etc.



<<Class Stereotype>>
ClassName

+publicAttribute: int

-privateMethod(param: int): void
#protectedMethod(): int
+abstractMethod(): void
+publicStaticMethod(): void

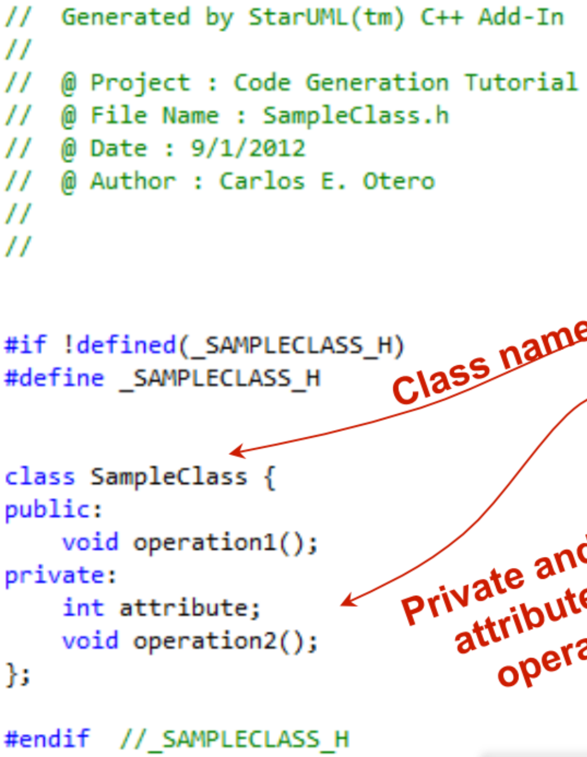
- Operation compartment
  - ✓ Reserved for the class' operations specification.
    - Including name, return type, parameters, visibility, etc.

- Everything specified in the UML class can be directly translated to code… let's see an example in the next slide…

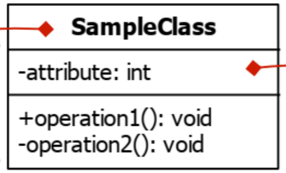
Class Diagram Revision

- ➢ Example of the *forward engineering* of a UML class to C++ and Java.

Code generated by free open source Star UML tool.



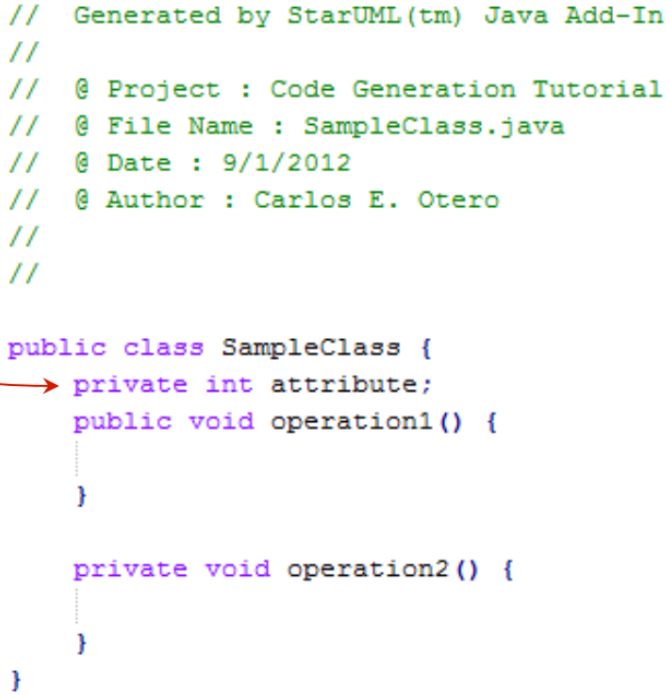
```
//  Generated by StarUML(tm) C++ Add-In
//
//  @ Project : Code Generation Tutorial
//  @ File Name : SampleClass.h
//  @ Date : 9/1/2012
//  @ Author : Carlos E. Otero
//
//


#if !defined(_SAMPLECLASS_H)
#define _SAMPLECLASS_H


class SampleClass {
public:
    void operation1();
private:
    int attribute;
    void operation2();
};

#endif  //_SAMPLECLASS_H
```



Class name

Private and public attributes and operations

Attribute name, type, and visibility

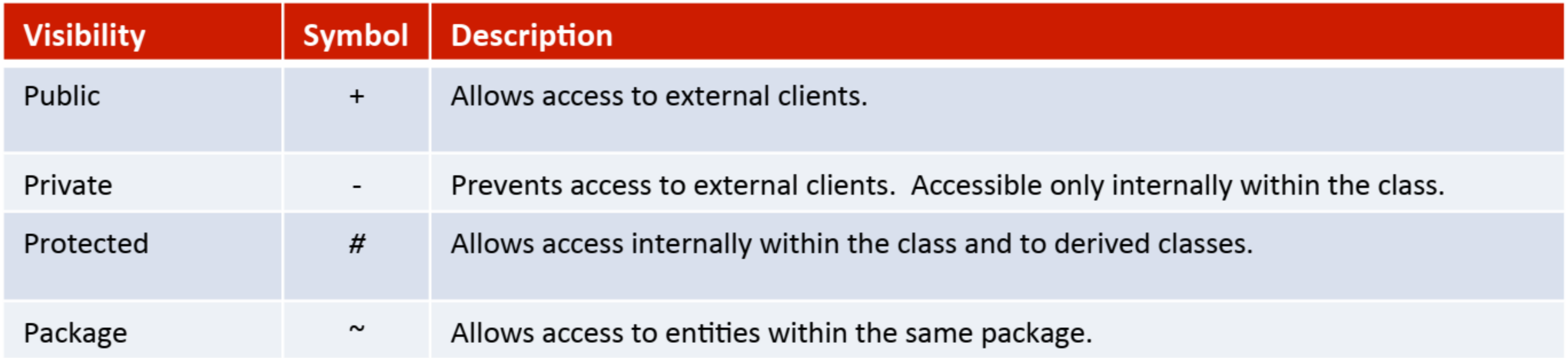


```
//  Generated by StarUML(tm) Java Add-In
//
//  @ Project : Code Generation Tutorial
//  @ File Name : SampleClass.java
//  @ Date : 9/1/2012
//  @ Author : Carlos E. Otero
//
//


public class SampleClass {
    private int attribute;
    public void operation1() {

    }

    private void operation2() {

    }
}
```

Important:
Notice how the modeled visibility {-, +} next to attribute and operations translate to code!

Class Diagram Revision

- In the previous slide, we presented two different types of UML visibility specification.
  - ✓ Visibility types specify policies on how attributes and operations are accessed by clients.
  - ✓ Common types of visibility are presented below.

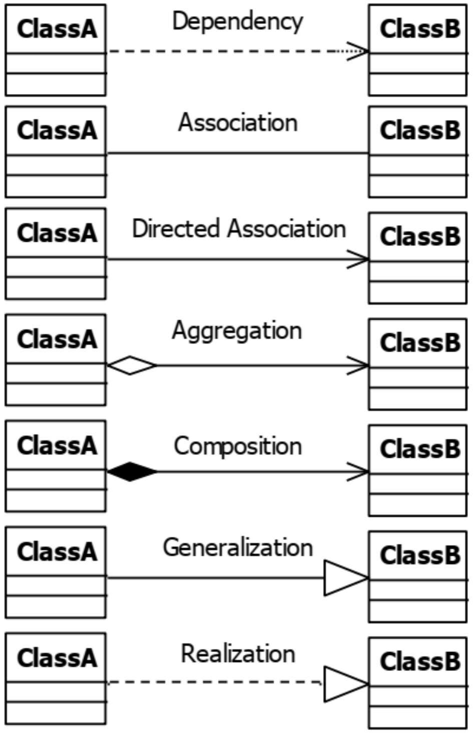


Visibility	Symbol	Description
Public	+	Allows access to external clients.
Private	-	Prevents access to external clients. Accessible only internally within the class.
Protected	#	Allows access internally within the class and to derived classes.
Package	~	Allows access to entities within the same package.

Important:
Visibility allows us to apply the Encapsulation principle in our designs!

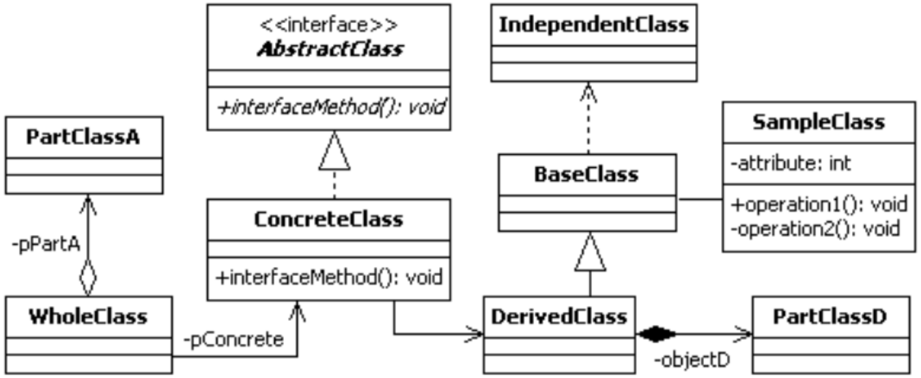
Class Diagram Revision

- UML relationships applied to the class classifier



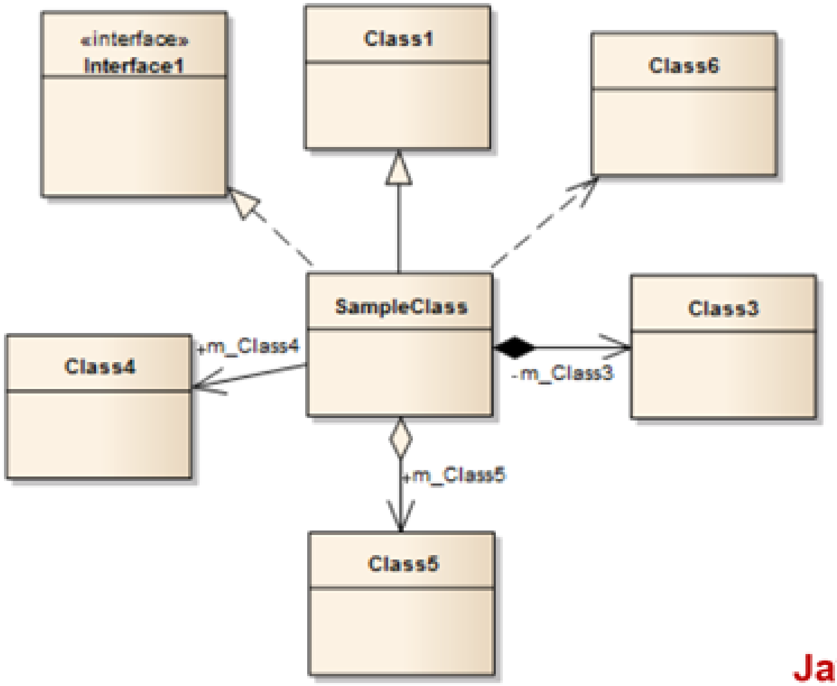
Important:
All of these relationships mean something in code, so that when you define these relationships, you're actually beginning to structure your code!

This is how a sample class diagram would look like

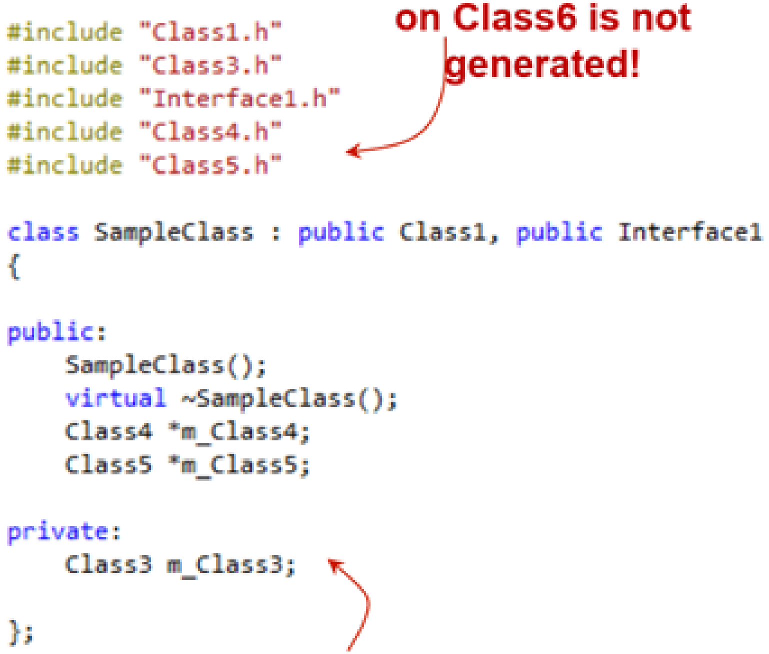


Class Diagram Revision

Model and Code generated by commercial Enterprise Architect UML tool.



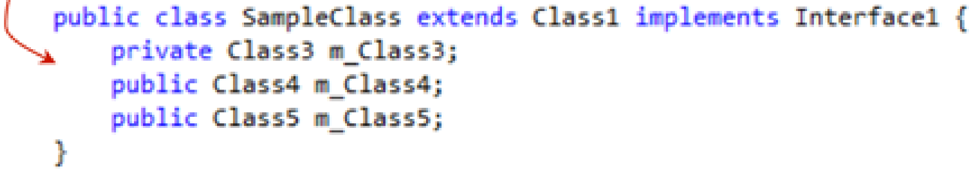
Notice that dependency on Class6 is not generated!



```
#include "Class1.h"
#include "Class3.h"
#include "Interface1.h"
#include "Class4.h"
#include "Class5.h"

class SampleClass : public Class1, public Interface1
{

public:
    SampleClass();
    virtual ~SampleClass();
    Class4 *m_Class4;
    Class5 *m_Class5;

private:
    Class3 m_Class3;

};
```

C++ code generation of model

Java code generation of same model



```
public class SampleClass extends Class1 implements Interface1 {
    private Class3 m_Class3;
    public Class4 m_Class4;
    public Class5 m_Class5;
}
```

Important:
Code generation varies from tool-to-tool. Some need to be configured appropriately to be useful in production environments!

“Users don’t know what they want until you show it to them”

Kent Beck

How to develop software which customer agrees with?

MOCKUP & PROTOTYPE

Prototyping tools

Adobe XD

Figma

InVision Studio

Sketch

Framer

Webflow

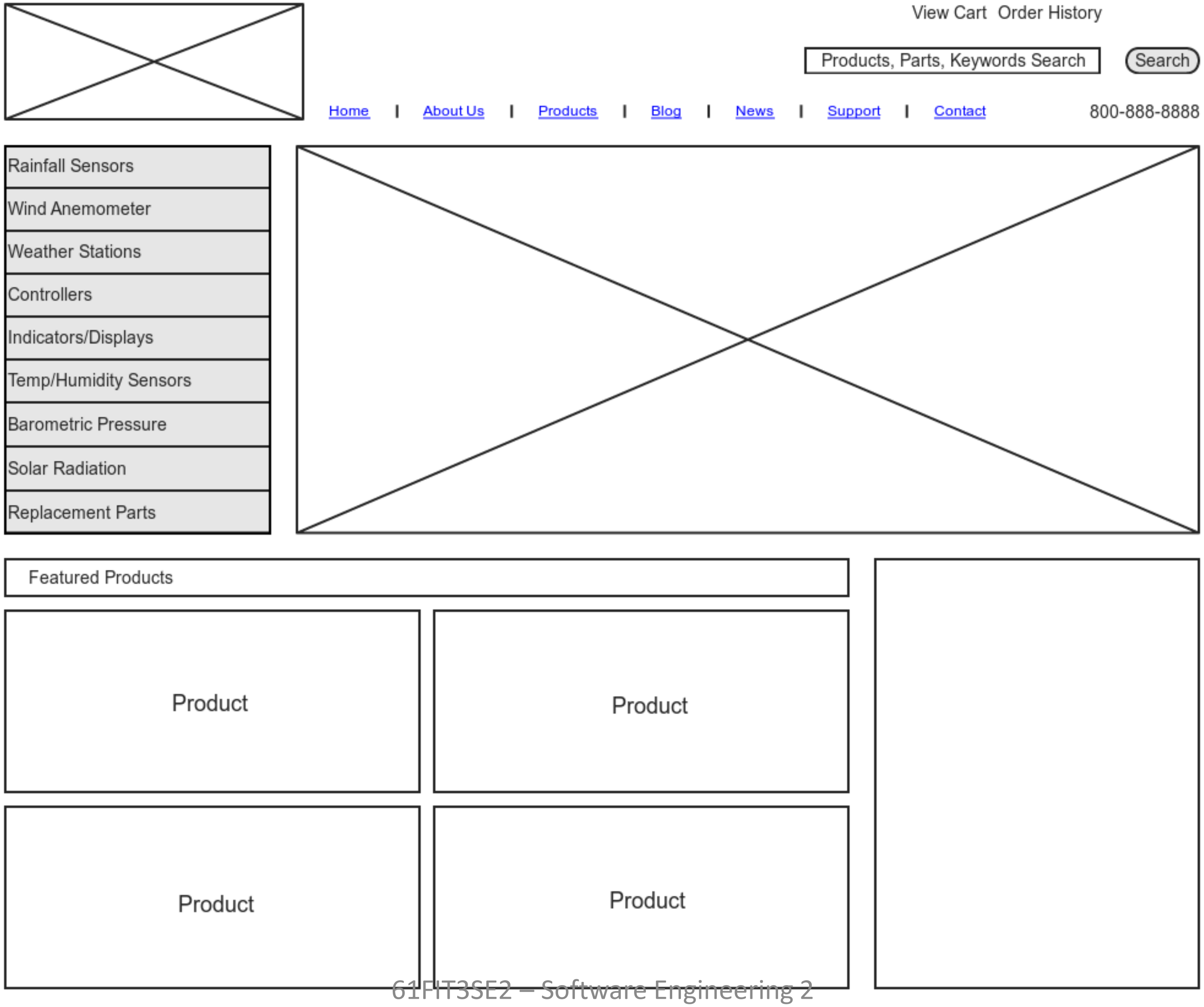
Prototyping tool features

- Graphical interface design
- Interactive prototype creation
  - Events, transitions, animations
- Team collaboration
- Reuse/Community
- Conversion from prototype to implementation
  - E.g. Convert to HTML/CSS

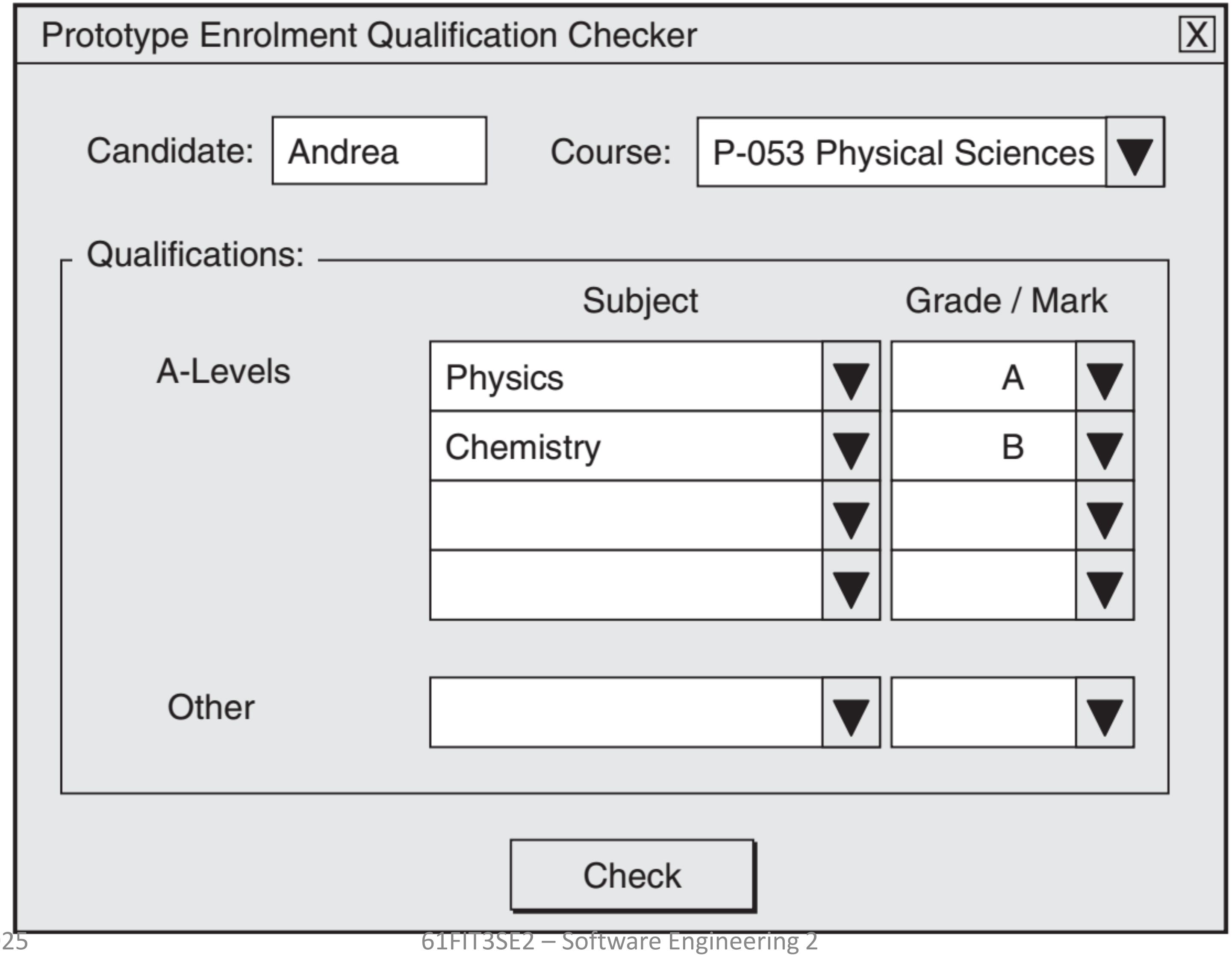
Prototype Fidelity

- **Wireframe:** a rough layout
- **Mockup:** draft version of UI using simple design elements
- **Prototype:** early version of the software that shows the UI design and is interactive

Example: Wireframe



Example: Mockup



Prototype Enrolment Qualification Checker

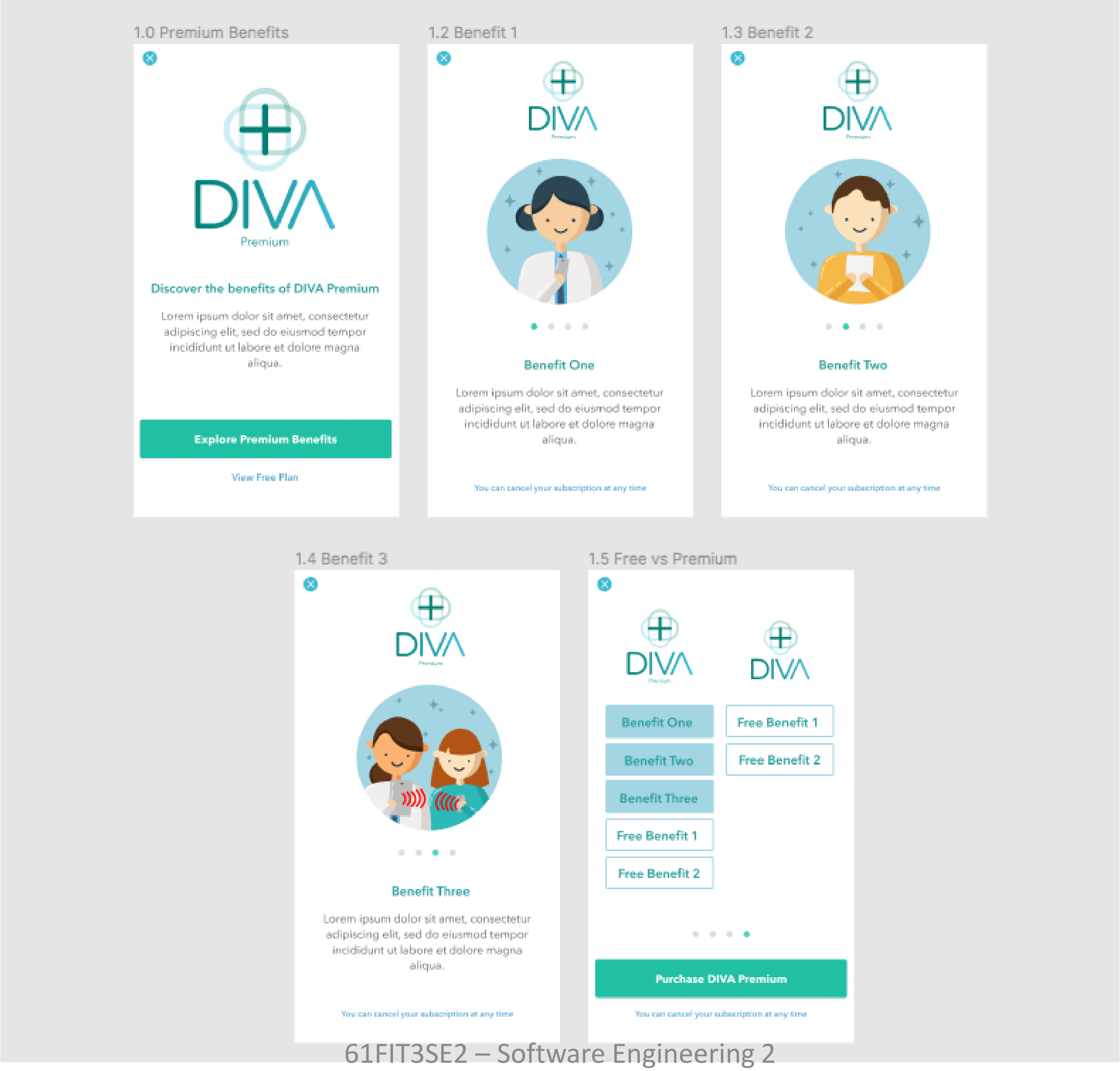
Candidate: Andrea

Course: P-053 Physical Sciences

Qualifications:

	Subject	Grade / Mark
A-Levels	Physics	A
	Chemistry	B
Other		

Example prototype



Discover the benefits of DIVA Premium

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

You can cancel your subscription at any time

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

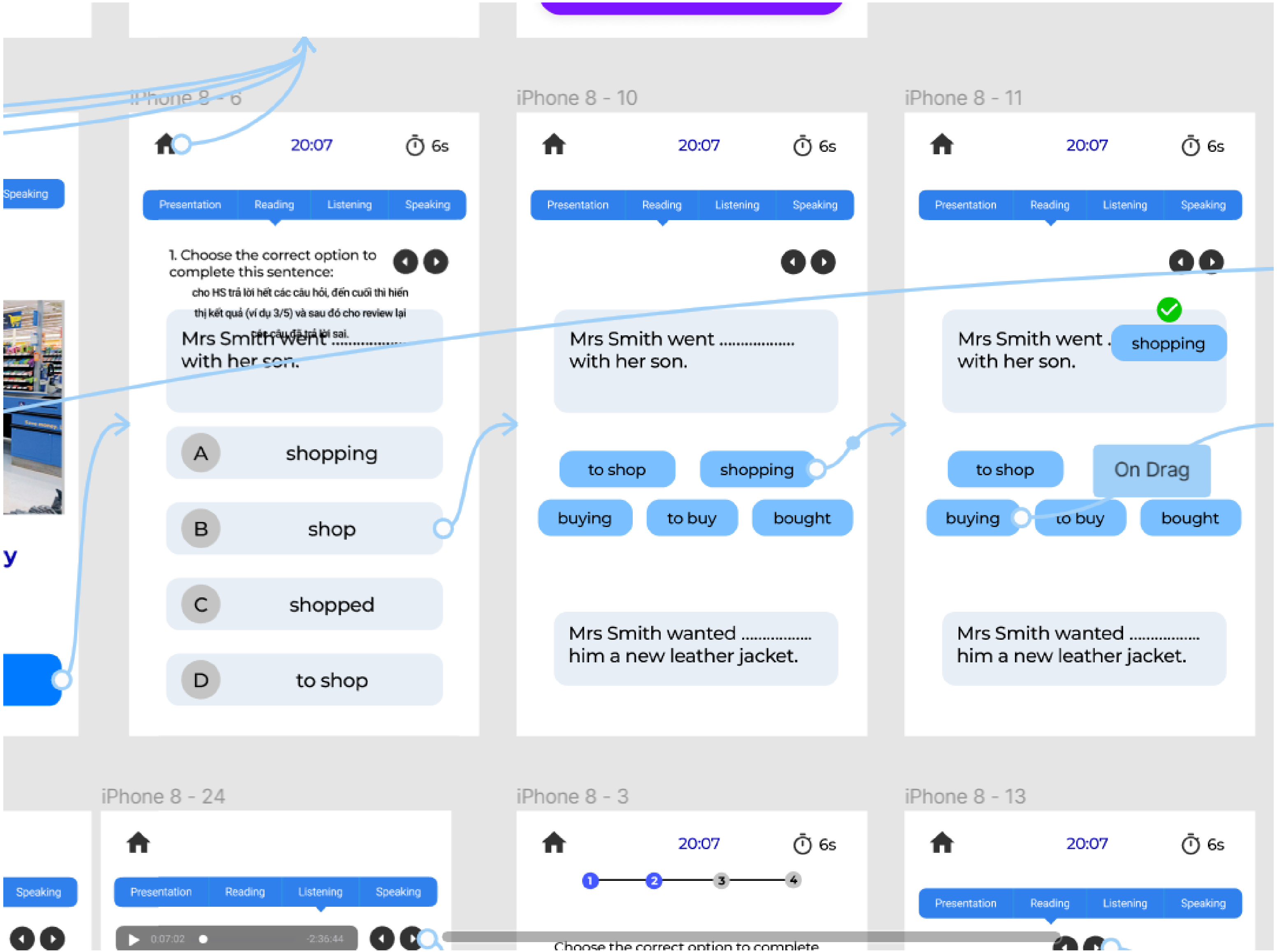
You can cancel your subscription at any time

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

You can cancel your subscription at any time

You can cancel your subscription at any time

Example Interactive Prototype



1. Choose the correct option to complete this sentence:

cho HS trả lời hết các câu hỏi, đến cuối thì hiển thị kết quả (ví dụ 3/5) và sau đó cho review lại các câu đã trả lời sai.

Mrs Smith went .... with her son.

Mrs Smith wanted .... him a new leather jacket.

Choose the correct option to complete

Spring Data JPA
Derived Query Methods

Spring Data JPA: Derived Query Methods

- By implementing `JpaRepository` interface, a repository will already have some basic CRUD methods (and queries) defined and implemented.
  - Built-in select queries: `findAll(), findById(), findAllById()`
  - Built-in insert/update queries: `save(), saveAll()`
  - Built-in delete queries: `delete(), deleteAll(), deleteById(), deleteAllById()`
- You can create more custom queries which are derived from method name.
- Next, you'll learn the naming convention for creating derived query methods.

Filtering data by a specific field

- If the `Employee` entity has a `name` field (and the standard `getName()` and `setName()` methods), we can define the `findByName()` method in `EmployeeRepository` interface.

```java
public interface EmployeeRepository
        extends JpaRepository<Employee, Long> {
    Employee findByName(String name);
}
```

- The correct query will be generated and implemented automatically.
  - It will be equivalent to the SQL query:
    ```
    select e from Employee e where e.name = ?1
    ```

Is / Equals vs. Like

- You can specify the *exact matching* operator after the attribute name. Examples:
  - `findByFirstname`, `findByFirstnameIs`, `findByFirstnameEquals`
    → … `where x.firstname = ?1`
  - `findByLastnameNot`
    → … `where x.lastname <> ?1`
  - `findByFirstnameLike`
    → … `where x.firstname like ?1`
  - `findByFirstnameNotLike`
    → … `where x.firstname not like ?1`

Containing, StartingWith, EndingWith

- You can specify the *partial matching* operator after the attribute name. Examples:
  - `findByNameContaining`
    → … `where x.name like ?1`
    (parameter bound wrapped in %)
  - `findByNameStartingWith`
    → … `where x.name like ?1`
    (parameter bound with appended %)
  - `findByNameEndingWith`
    → … `where x.name like ?1`
    (parameter bound with prepended %)

The `And` and `Or` keywords

- `Employee findByNameOrAddress(String n, String addr)`
  - … `where x.name = ?1 or x.address = ?2`
- `Employee findByNameContainingAndAge(String n, int age)`
  - … `where x.name like ?1 and x.age = ?2` (`name` parameter wrapped in %)

Some other keywords

- ✧ Querying distinct records:

```
Employee findDistinctByName(String n)
```

```
→select distinct … where x.name = ?1
```

- ✧ Ignoring character case when matching value:

```
Employee findByNameIgnoreCase(String n)
```

```
→ … where UPPER(x.firstname) = UPPER(?1)
```

JPA Criteria API

Why use Criteria API?

- ✧ Most applications provide a front end for users to search for information.
  - Typically, many searchable fields are displayed, and the users enter information in only some of them and do the search.
- ✧ It's difficult to prepare many queries, with each possible combination of parameters that users may choose to enter.
- ✧ The Criteria API query feature is a solution to this problem.

Basic concepts of the Criteria API

- ✧ **CriteriaBuilder**: Used to construct criteria queries, compound selections, expressions, predicates, and ordering.
- ✧ **CriteriaQuery**: Represents a query object.
- ✧ **Root**: Represents the entity in the FROM clause.

How to query with Criteria API?

- Let’s start with a simple example:
    - `SELECT * FROM person`
- With Criteria API, you would write:

```
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaQuery<Person> cq = cb.createQuery(Person.class);
Root<Person> root = cq.from(Person.class);
cq.select(root);

List<Person> results = entityManager.createQuery(cq).getResultList();
```

How to add WHERE clause to Criteria Query?

- You need to use Predicate(s)
- In the Criteria API, a predicate represents a condition or a filter that you apply to your query.
  - Think of it as a way to specify the criteria that the data must meet

```
// age >= 18
Predicate agePredicate = cb.greaterThan(root.get("age"), 18);
// name LIKE 'John%'
Predicate namePredicate = cb.like(root.get("name"), "John%");
// Combine predicates using AND
Predicate finalPredicate = cb.and(agePredicate, namePredicate);
// Apply the predicates to the query
cq.where(finalPredicate);
```

How to combine multiple predicates?

✧ When combining more than 2 predicates using the AND operator, use the following syntax:

```
// Create an array of predicates
Predicate[] predicates = new Predicate[3];
predicates[0] = cb.greaterThan(root.get("age"), 18);
predicates[1] = cb.like(root.get("name"), "John%");
predicates[2] = cb.like(root.get("male"), true);
// Apply the predicates to the query using the AND operator
cq.where(predicates);
```

How to sort in Criteria Query?

- Use the `CriteriaBuilder` to create an `Order` object, which specifies the sorting order (ascending or descending).
- Apply this `Order` object to the query.

```java
// Obtain an instance of CriteriaBuilder
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
// Create a query object for the Person entity
CriteriaQuery<Person> cq = cb.createQuery(Person.class);
// Define the root of the query (the FROM clause)
Root<Person> root = cq.from(Person.class);
// Create an Order object for sorting by age in ascending order
Order order = cb.asc(root.get("age"));
// Apply the order to the query

cq.orderBy(order);
```

Getting and using Criteria Query result

```java
// Execute the query and get the results
List<Person> results = entityManager.createQuery(cq).getResultList();

// Process the results
for (Person person : results) {
    System.out.println("Name: " + person.getName() +
            ", Age: " + person.getAge());

}
```