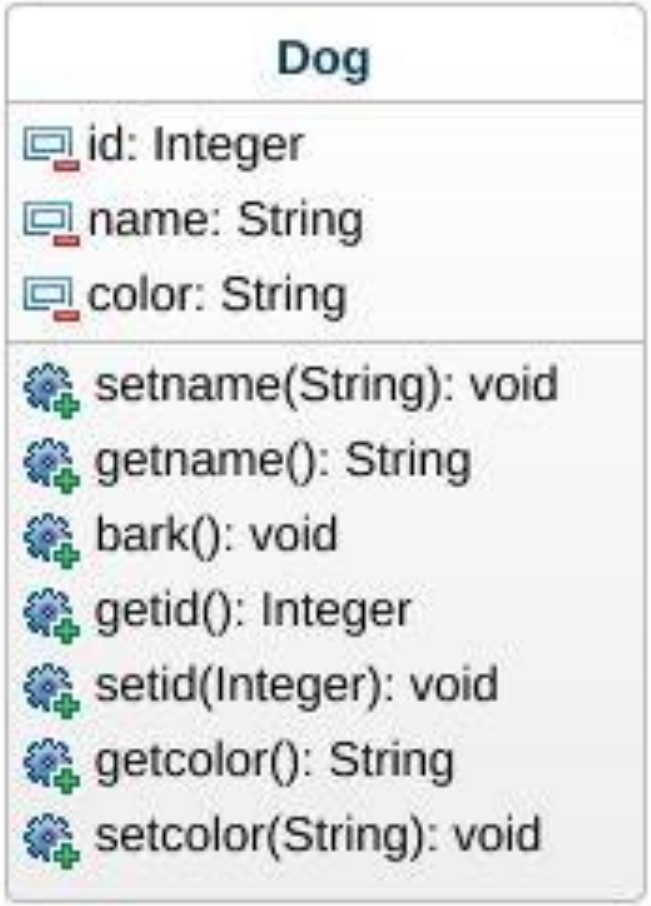
Object Oriented Design Implementation

Object Oriented Implementation

- With the completion of interaction diagrams & DCDs, there is sufficient detail to generate code for domain layer of objects.
- The UML artifacts created during the design work(Interaction diagram & DCD) will be used as input to the code generation process.
- The implementation Model in UP contains the implementation artifacts such as source code, database definition, JSP/XML/HTML pages, &so forth.
- Having wrapped up with design issues with visibility, this chapter introduces mapping of design to code in OO language.

Mapping Single Class to Code

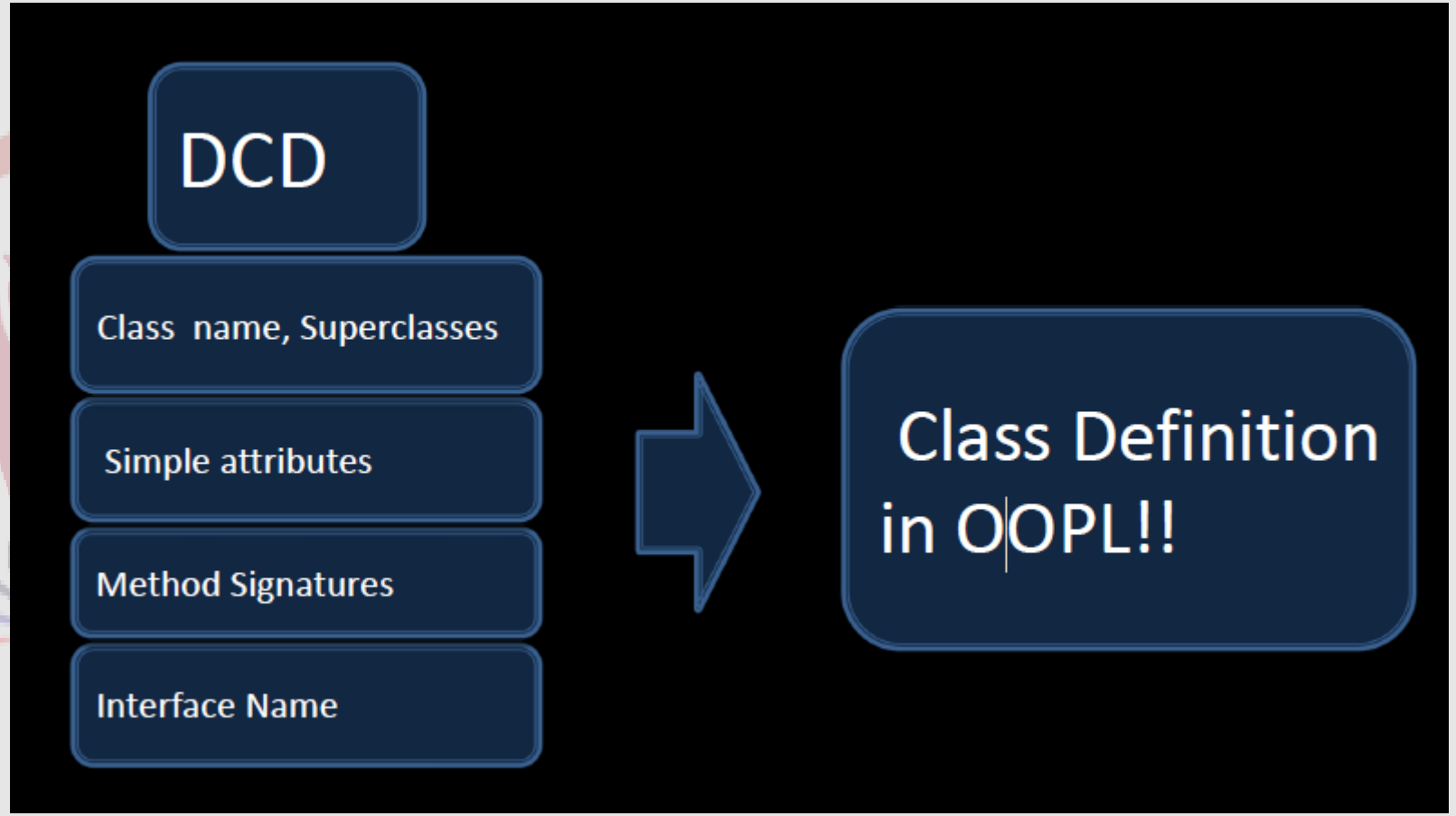



```
public class Dog {
//attributes:
        private Integer id;
        private String name;
        private String color;
//operations:
        public Integer getId() {
        return this.id;
    }
        public void setId(Integer id) {
        this.id = id;
    }

        public String getName() {
        return this.name;
    }
       public void setName(String name) {
        this.name = name;
    }
        public String getColor() {
        return this.color;
    }

        public void setColor(String color) {
        this.color = color;
    }

       public bark() {
        .....}
        }
```

Creating class definitions from DCDs

- At the very least, DCDs depict the class or interface name, superclasses, operation signatures, and attributes of a class.
- This is sufficient to create a basic class definition in an OO language.
- If the DCD was drawn in a UML tool, it can generate the basic class definition from the diagrams.

Creating class definitions from DCDs



Defining a Class with Methods and Simple Attributes

From the DCD,

- a mapping to the attribute definitions(java fields) and
- method signatures for the Java definition of SalesLineItem is straightforward as shown in fig below.
- The create method is often excluded from the class diagram because of its commonality and multiple interpretations, depending on the target language.

Interface

- Interface is the collection of abstract methods: A method which is declared but can’t be defined called abstract method
- Using interface, you can specify what a class must do, but not how it does it.
- It is not a class but a set of requirements for classes that implement the interface
- A class implements and interface thereby inheriting the abstract method of interface

Real Life Example of Interface

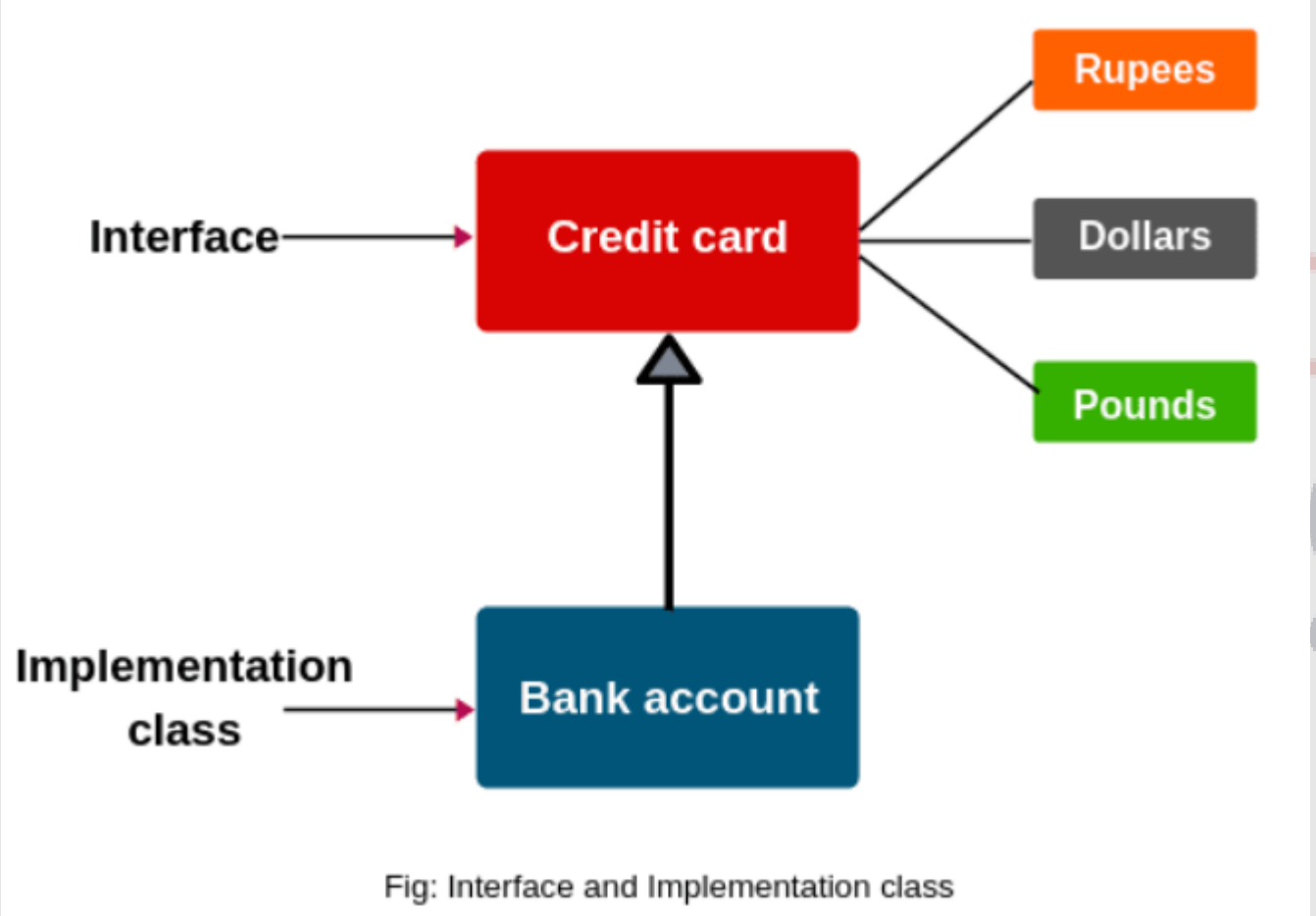


Fig: Interface and Implementation class

Mapping Class Diagrams to java

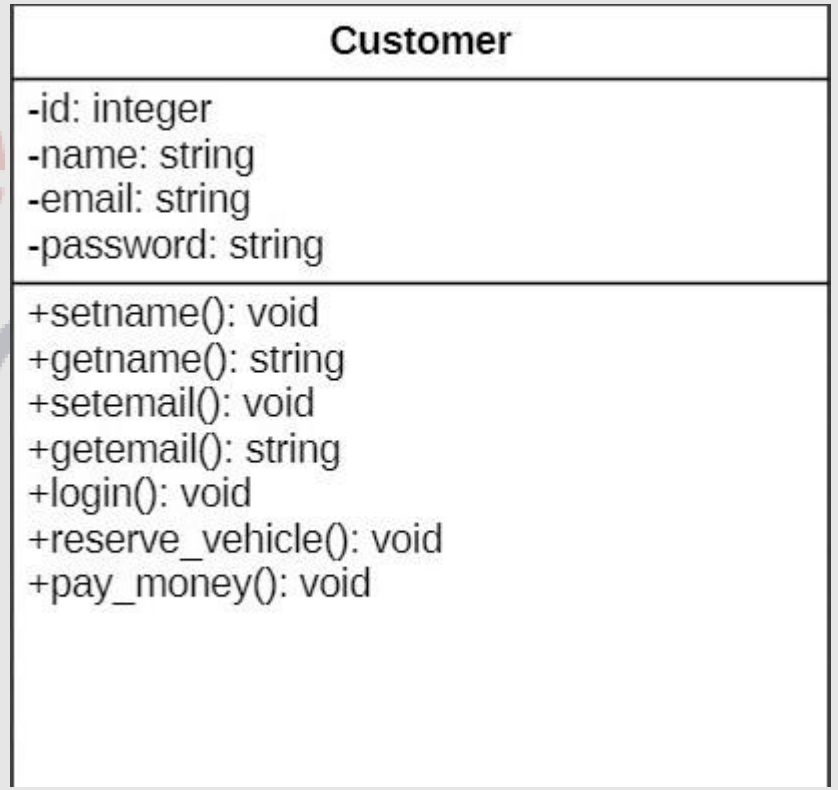
Things to be considered to map Class Diagrams into code:

1. Class should be defined without varying the class name
2. All the Attributes of each classes must be mapped with the same name and attribute type
3. All the Functions should be mapped with the same function name and function type
4. Relationships between the classes must be visualized simply in the code including multiplicity.
5. A complete structure of the class diagram must be mapped into the java code(Class Diagram being a structural diagram, prioritize the structural part more)
6. The mapped code must provide a structure of the class diagram of the system containing all the contents of the classes

1. Mapping Class name

Creating Class Definition from Design Class Diagram

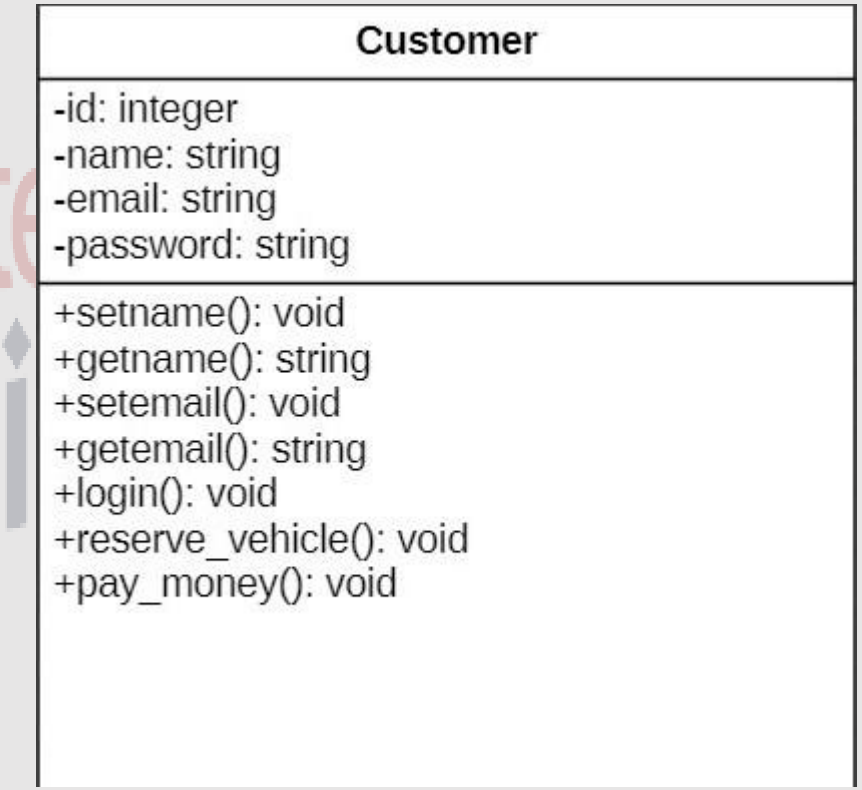
```
Public Class Customer
{
//define attributes
//define functions
}
```



2. Map All Attributes of each class

Updating Class Definitions by adding attributes

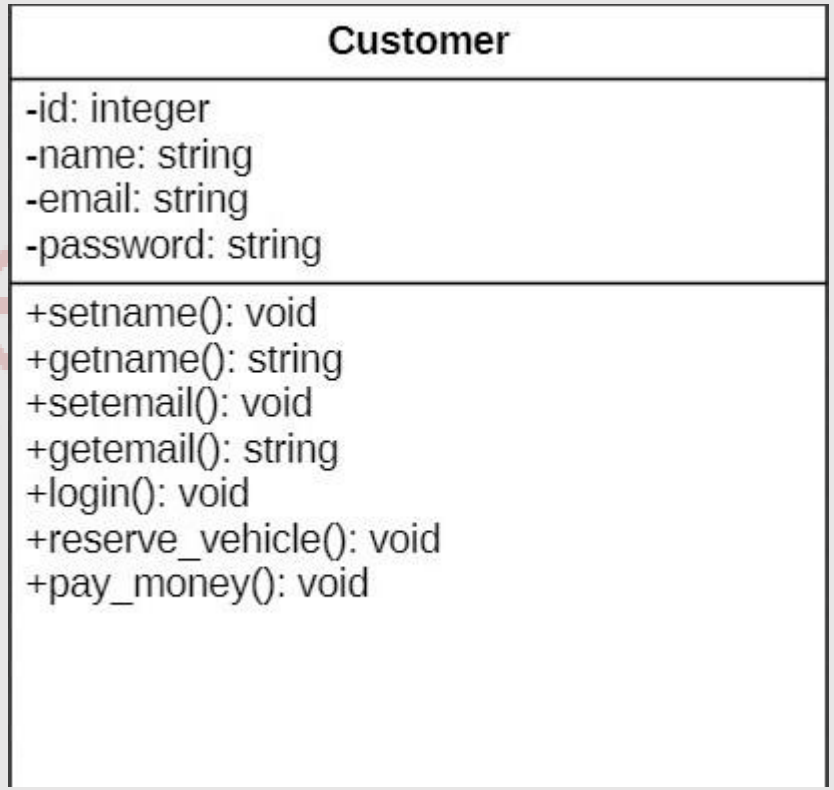
```
Public Class Customer
{
Private integer id;
Private string name;
Private string email;
Private string password;
//define functions
}
```



3. Map All functions of each class

Updating Class Definitions by adding functions

```
Public Class Customer
{
//define attributes
Public void setname() {….}
Public string getname() {….}
Public void setemail() {….}
Public string getemail() {….}
Public void login() {….}
Public void reserve_vehicle() {….}
Public void pay_money() {….}
}
```



4.Relationships in Class Diagrams

Updating Class Definition by adding relationships

Association Relationship:

- The association represents the static relationship between two classes along with the multiplicity.
- E.g. an employee can have one primary address associated with it but can have multiple mobile numbers.
- Multiplicity defines how many instances can be associated at any given moment.
- One to one association
- One to many association
- Many to many association

Exception and Error Handling

- An exception is a condition that is caused by a runtime error in the program.
- It is a situation in which a program has an unexpected behavior that the section of code containing the problem is not designed to handle.
- It is a problem that arises during the execution of a program.
- An exception can occur for many different reasons, including the following:
  - ✓A user has entered invalid data.
  - ✓A file that needs to be opened cannot be found.
  - ✓A network connection has been lost in the middle of communications, or the JVM((Java Virtual Machine) has run out of memory.
- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Categories of exceptions

- **Checked exceptions:**
  - ✓ It is an exception that is typically a user error or a problem that cannot be foreseen by the programmer.
  - ✓ For example, if a file is to be opened, but the file cannot be found, an exception occurs.
  - ✓ These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:**
  - ✓ It is an exception that occurs that probably could have been avoided by the programmer.
  - ✓ As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
- **Errors:**
  - ✓ These are not exceptions at all, but problems that arise beyond the control of the user or the programmer.
  - ✓ Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise.
  - ✓ They are also ignored at the time of compilation.

Exception Handling Process

- A method can signal an error condition by throwing an exception – throws
- The calling method can transfer control to a exception handler by catching an exception - try, catch
- Clean up can be done by – finally
  - Try block, code that could have exceptions errors
  - Catch block(s), specify code to handle various types of exceptions. First block to have appropriate type of exception is invoked.
  - If no 'local' catch found, exception propagates up the method call stack, all the way to main()
  - Any execution of try, normal completion, or catch then transfers control on to finally block

Exception Handling Process

```
class WithExceptionHandling{
        public static void main(String[] args){
                int a,b;  float r;
                a = 7;   b = 0;
                try{
                        r = a/b;
                        System.out.println("Result is " + r);
                }
                catch(ArithmeticException e){
                        System.out.println(" B is zero);
                }
                System.out.println("Program reached this line");
        }
}
```

Program Reaches here

Fig: Example of Exception Handling

```
class WithExceptionCatchThrowFinally{
        public static void main(String[] args){
                int a,b;  float r;  a = 7;   b = 0;
                try{
                        r = a/b;
                        System.out.println("Result is " + r);
                }
                catch(ArithmeticException e){
                        System.out.println(" B is zero);
                        throw e;
                }
                finally{
                 System.out.println("Program is complete");
                }
        }
}
```

Program reaches here

Fig: Example of Exception Handling with finely

Summary of Exception Handling

- A good programs does not produce unexpected results.
- It is always a good practice to check for potential problem spots in programs and guard against program failures.
- Exceptions are mainly used to deal with runtime errors.
- Exceptions also aid in debugging programs.
- Exception handling mechanisms can effectively used to locate the type and place of errors.