

BACS2063 Data Structures and Algorithms

# Abstract Data Types (ADTs)

Chapter 2

# Learning Outcomes

At the end of this lecture, you should be able to

- Explain the **benefits** of abstract data types (**ADTs**).
- **Write** **ADT** specifications.
- ~~**Implement** **ADTs** using Java interfaces and classes.~~

# Winning Strategies in Programming

## Increase Productivity

- Projects can be finished **on time**
- **More** projects can be handled

## Assure Quality

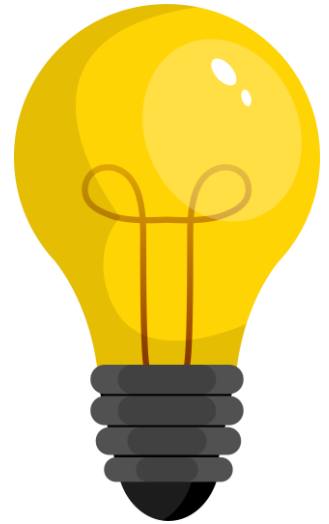
- Reliable: **bug-free**

How to ... *increase productivity*  
and *assure quality*?

Code reuse

Code maintainability

perfective  
adaptive  
corrective



For reading only, not included in the syllabus

# **3 TYPES OF MAINTENANCE (APPENDIX)**

# To achieve *reuse*

*free from language type*

*design*  
Abstraction

- ADT specification

*implementation coding*  
Encapsulation

- ADT implementation

# To achieve *maintainability*

Encapsulation  
/ Information  
Hiding

- ADT implementation

# Benefits of

## Abstraction

- Can focus on the abstract properties without worrying about how it is going to be implemented.

## Encapsulation

- Can use the components without knowing the implementation details.



# 2020 October, Q1c



- c) Analyze **TWO (2)** benefits of using abstraction in the data structures. Justify your answer with relevant examples. (8 marks)

## Answers:

The main benefits of using abstraction in data structures are:

### **Reusability**

*By defining the ADT and creating its implementation with Generic data type, a particular ADT can later be used in any system that needs a collection of data which has the same characteristics as the ADT.*

### **Maintainability**

*The definition of the ADT and its implementation are separate, so if the implementation needs to be changed, the users of the ADT do not need to change their code. Also, ADT is written in modules, so any code modifications are easy to be maintained.*

# Abstraction & Encapsulation

Abstraction

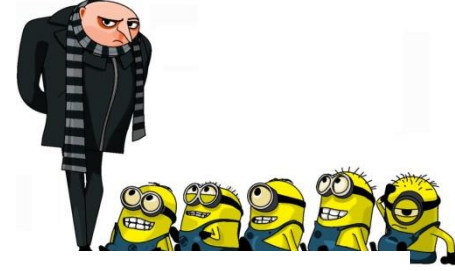
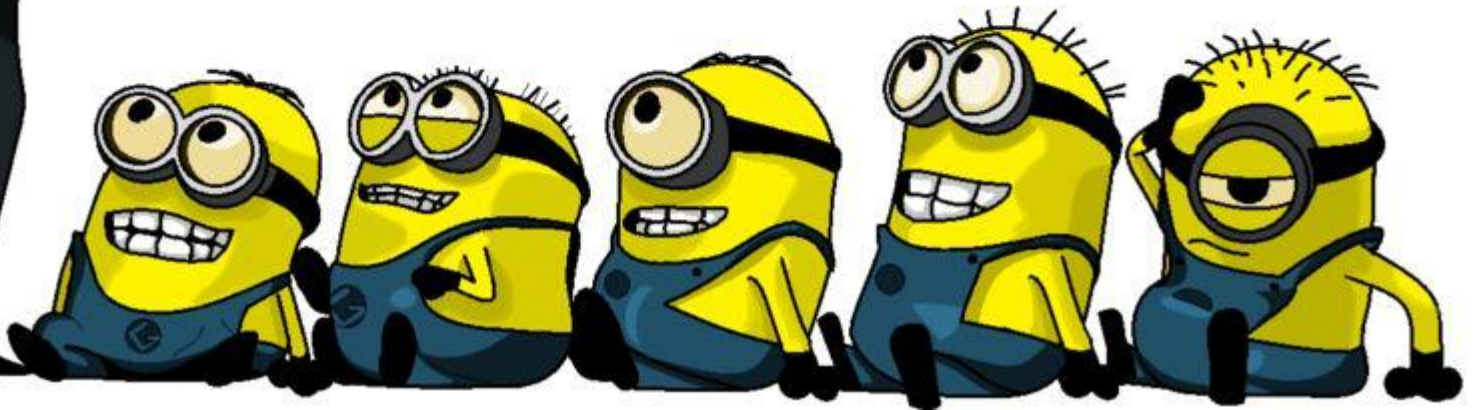
- Specifying the data type

Encapsulation

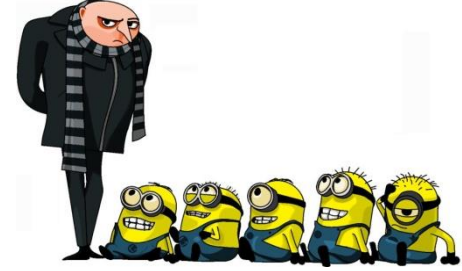
- Using the data type

**2 sides of the same coin**

# Case Study



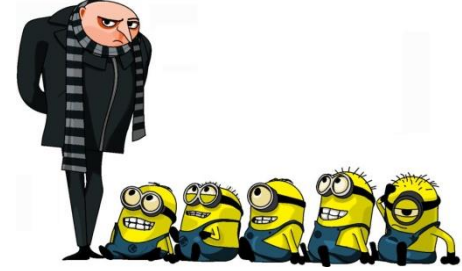
# Case Study



Mr. Gru set up MinionSoft in January 2013. Since then, his company has successfully completed 7 projects. As Gru reflected on his projects, he suddenly realized a similarity in all his projects – each of them used a *list* of some sort. *E.g.*,

- The BananaBananana MP5 player software had a *playlist for songs*.
- In the MonsterMinion game, each character had a *weapon list*.
- The OhPotato! productivity app had a *task list*.

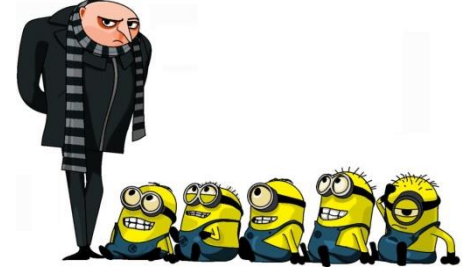
# Case Study



Mr. Gru set up MinionSoft in January 2013. Since then, his company has successfully completed 7 projects. As Gru reflected on his projects, he suddenly realized a similarity in all his projects – each of them used a *list* of some sort. *E.g.*,

- The BananaBananana MP5 player software had a *playlist for songs*.
- In the MonsterMinion game, each character had a *weapon list*.
- The OhPotato! productivity app had a *task list*.

# Case Study



Mr. Gru set up MinionSoft in January 2013. Since then, his company has successfully completed 7 projects. As Gru reflected on his projects, he suddenly realized a similarity in all his projects – each of them used a *list* of some sort. *E.g.*,

- The BananaBananana MP5 player software had a *playlist for songs*.
- In the MonsterMinion game, each character had a *weapon list*.
- The OhPotato! productivity app had a *task list*.

# Gru's observation

- Code duplication – reinventing the wheel
- Not-so-maintainable code

# MinionSoft's code duplication

- The list is a “thing” that appears in all of the software applications.
- It has specific characteristics
  - Data in the list is organized in a certain way
  - There are certain operations that are performed on the list
- The declaration for the array to hold the list elements and the coding for the operations were repeated in all the software applications.



# MinionSoft's not-so-maintainable code

- If a bug is discovered, the changes to the code would need to be applied to every single software module or application which used a list.

# How to...

- Enable reuse?
- Increase maintainability?

# Solution Steps

## 1. Abstraction

- Identify the **general/abstract properties** and **operations** of the list.
  - An abstract data type
- Produce a **specification** of the list
  - What are the characteristics of the data?
  - What are the operations for manipulating the data?

# Solution Steps (cont'd)

## 2. Encapsulation

- Implement the list in such a way that a programmer can use the list without knowing how it is implemented.

### ➤ Information hiding



## ❖ An *ADT* is

“An abstraction of a commonly appearing data structure along with a set of defined operations on the data structure. It specifies the logical properties *without the implementation details*. In Java, ADT is achieved through the use of classes.”

Malik & Nair (2003)

# ADT Specifications are

- Written in a **natural language** (e.g. English) and are **independent of any programming language**.
- Used as specifications for concrete data types (i.e. the actual data types used in programs).

# What to include in an ADT specification?

1. **ADT title**
2. **Description** of the characteristics (logical properties) of the data type
3. Description of each **operation**:
  - **Operation header**: return type (if any), operation name, parameters (if any)
  - **Brief description** of what the operation does
  - **Precondition** (*if any, i.e. this is optional*)
  - **Postcondition**
  - What is **returned** by the operation (*if any, i.e. this is optional*)

# Preconditions and Postconditions

## Precondition

- A statement specifying the condition(s) that must be **true** before the operation is invoked.

## Postcondition

- A statement specifying what is **true** after the operation is completed.

– Malik & Nair (2003)



# PYQ 19-Jan-2023 (Question)



## Question 1

Figure 1 shows the formula for area of various shapes. Consider an Abstract Data Type (ADT) called **AreaShape** which represents area of a shape or size of the shape.

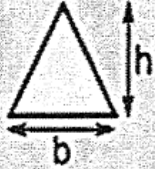
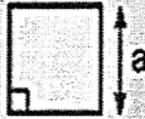
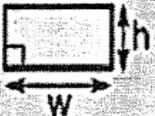

	<p><u>Triangle</u></p> <p>Area = <math>\frac{1}{2} \times b \times h</math> b = base h = vertical height</p>		<p><u>Square</u></p> <p>Area = <math>a^2</math> a = length of side</p>
	<p><u>Rectangle</u></p> <p>Area = <math>w \times h</math> w = width h = height</p>		<p><u>Circle</u></p> <p>Area = <math>\pi \times r^2</math> Circumference = <math>2 \times \pi \times r</math> r = radius</p>

Figure 1: Area of shapes

ADT Title: **AreaShape**

Area of a shape is the size of the surface.

Description of each operation as below:

- *triangleArea*(b, h), which calculate the area of a triangle
- *rectangleArea*(w, h), which calculate the area of a rectangle
- *squareArea*(a), which calculate the area of a square
- *circleArea*(r), which calculate the area of a circle

- a) Write an ADT specification named **AreaShape** which include operations that mentioned above and state any assumptions made. (10 marks)

# Answer:

## ADT **AreaShape**

Area of a shape is the size of the surface.

Integer *triangleArea* (Integer base, Integer height )

Description : calculate the area of a triangle

Pre-condition : must be integer value

Post-condition : area of triangle calculated

Return : result of area calculation

Integer *rectangleArea* (Integer width Integer height )

Description : calculate the area of a rectangle

Pre-condition : must be integer value

Post-condition : area of rectangle calculated

Return : result of area calculation

Integer *squareArea* (Integer width )

Description : calculate the area of a square

Pre-condition : must be integer value

Post-condition : area of square calculated

Return : result of area calculation

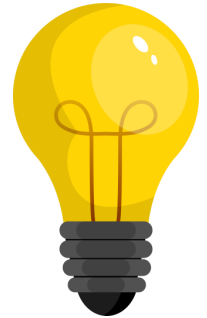
double *circleArea* (Integer radius )

Description : calculate the area of a circle

Pre-condition : must be integer value

Post-condition : area of circle calculated

Return : result of area calculation



# PYQ 8-June-2023 (Question)



## Question 1

Figure 1 shows a sample output screen of the Meter conversion. Consider an Abstract Data Type (ADT) called **Converter** which converts a unit of meter to miles, meter to feet and meter to inches.

```
Please enter a Measurement in Meters: 50

The conversion measurement is as follows
Miles   : 0.0311
Feet    : 164.0420
Inches  : 1968.5039
```

Figure 1: Sample output screen of meter conversion application

### ADT Converter

Converter is a conversion that converts meters to miles, feet and inches.

Description of each operation as below:

- *miles* (m), which convert meters to miles
- *feet* (m), which convert meters to feet
- *inches*(m), which convert meters to inches

- a) Write an ADT specification named **Converter** which, includes the above-mentioned operations. State any assumptions made. (10 marks)

# Answer

## ADT Converter

Converter is a conversion that convert meters to miles, feet and inches.

### **Double *miles* (Integer meter)**

Description : convert given meter value to miles

Pre-condition : must be integer value

Post-condition : given meter value remain unchanged

Return : miles value returned

### **Double *feet* (Integer meter)**

Description : convert given meter value to feet

Pre-condition : must be integer value

Post-condition : given meter value remain unchanged

Return : feet value returned

### **Double *inches* (Integer meter )**

Description : convert given meter value to inches

Pre-condition : must be integer value

Post-condition : given meter value remain unchanged

Return : inches value returned



# An ADT Specification must be *generic* in nature



- An ADT specification must not indicate what **programming language** would be used to implement it.
- An ADT specification for a **collection ADT** must be generic in nature and **free from these 2 things**:
  - **Implementation details** (e.g. using **array** or **linked nodes**)
  - **Detail of the specific objects contained** (e.g **Product** objects, or **Customer** objects, etc.)
- **For Example**, for the ADT specification of a List ADT, you must not indicate whether the List will be implemented with **array** or with **linked nodes**. You must also not specify that this List will contain **Customer** objects or **Product** objects.

# Words you should not use in an ADT Specification (1/2)



- Examples of words that you **can** use
  - List
  - Stack
  - Queue
- Examples of words that you **cannot** use
  - Array List / Linked List
  - Array Stack / Linked Stack
  - Array Queue / Linked Queue

# Words you should not use in an ADT Specification (2/2)



- For example, here are 3 method definitions:
  1. `add(T newEntry)` ← CORRECT
  2. `add(Customer newEntry)` ← WRONG
  3. `add(Product newEntry)` ← WRONG
- The add method defines the add operation of an ADT. The object to be added is received as a parameter. This object should be generic to allow reusability. Reusability means I can use this ADT in Project X to store *customer* objects. I can also use this ADT in Project Y to store *product* objects.
- The first one is done correctly, because the parameter is of a generic type.
- The second & third ones are wrong and should not be done in an ADT specification because *Customer* and *Product* are specific types and not generic. When the object is specific, it no longer allow reusability.

# ADT Specification and New Data Structure



- **Purpose of creating an ADT Specification**
  - To document the specifications of a new data structure in a standard way and is generic, and allow this specification to be reused and **implemented in any programming languages**.
- **Characteristics of a new data structure**
  - Based on the ADT specification, a new data structure will be created and implemented (with a programming language) which is done with **encapsulation** so that the **implementation details** are kept **hidden** from the user.



# Sample for common mistakes (1)



## 2. Abstract Data Type (ADT) Specification

### ADT ArrayList

An **array** list is a resizable list that is used to store the elements. It will grow its size to accommodate the new elements and it will shrink the size when the elements are removed. It also allows retrieval of the elements by their index.

<b>Boolean insert(T newData)</b>	
Description	Add <b>newData</b> to <b>the list</b> .
Precondition	-
Postcondition	The <b>newData</b> is added to the list.
Return	Return <i>true</i> if the <b>newData</b> is successfully added to the list, else return <i>false</i> .

# Sample for common mistakes (2)



## 2. Abstract Data Type (ADT) Specification

ADT: List

Implementation method: Array List

Boolean add(T newEntry)	
Description	Add newEntry into the list
Precondition	-
Postcondition	newEntry has been added to the end of the list
Returns	True if newEntry is successfully added

# Sample for common mistakes (3)



## 2. Abstract Data Type (ADT) Specification

### ADT `LinkedListDoubly`

A `LinkedListDoubly` is used to make some operations such as add, remove and get number of entries etc.

boolean addFirst(T newEntry)	
Description	Add newEntry to the first place of the <code>doubly linked</code> list.
Postcondition	newEntry has been added to the <code>doubly linked</code> list.
Returns	True if newEntry successfully added, false if not.

boolean addLast(T newEntry)	
Description	Add newEntry to the last place of the <code>doubly linked</code> list.
Postcondition	newEntry has been added to the <code>doubly linked</code> list.
Returns	True if newEntry successfully added, false if not.

# Sample for common mistakes (4)



## 2. Abstract Data **type** (ADT) Specification

enqueue(T newEntry)	
Description	adds a newEntry <b>node</b> to after rear and moves the rear to the <b>next node</b> .
Pre Condition	-
Post Condition	newEntry has been added to the <b>rear Node</b> of the queue
Returns	-

# Sample for common mistakes (5)



## 2. Abstract Data Type (ADT) Specification

ADT: Queue

Implementation method: **Circular Linked Queue**

void enqueue (T newEntry)	
Description	Adds new entry at the back of the queue
Pre Condition	-
Post Condition	New entry is added to the queue
Returns	-



# Sample for ADT Specs done correctly

<b>Boolean add (T newElement)</b>	
<b>Description</b>	Add newElement to the end of the list
<b>Pre-condition</b>	-
<b>Post-condition</b>	The newElement is added to the end of the list
<b>Returns</b>	Return true if newElement successfully added, else return false

# Problem: A Counter ADT

- **Counter** devices are used for counting things such as cars entering a parking lot, people taking numbers at the post-office, etc. A counter object would have a **non-negative integer value** representing the **current count**. It can be *incremented*, *decremented*, *reset* to zero and have its value *read* at any time.

# Exercise 2.1 (Answer)



- Write the ADT specification for a counter whose instances would represent counter objects.

## ADT Counter

A counter is an object for counting things. It has a non-negative integer value representing the current count.

### increment()

<b>Description</b>	Increment the current count by 1.
<b>Postcondition</b>	This object's count has been incremented by 1.

### decrement()

<b>Description</b>	Decrement the current count by 1.
<b>Precondition</b>	The current count is more than 0.
<b>Postcondition</b>	The object's count has been decremented by 1.

### reset()

<b>Description</b>	Reset the value of this counter to 0.
<b>Postcondition</b>	This object's value has been changed to 0.

### Integer read()

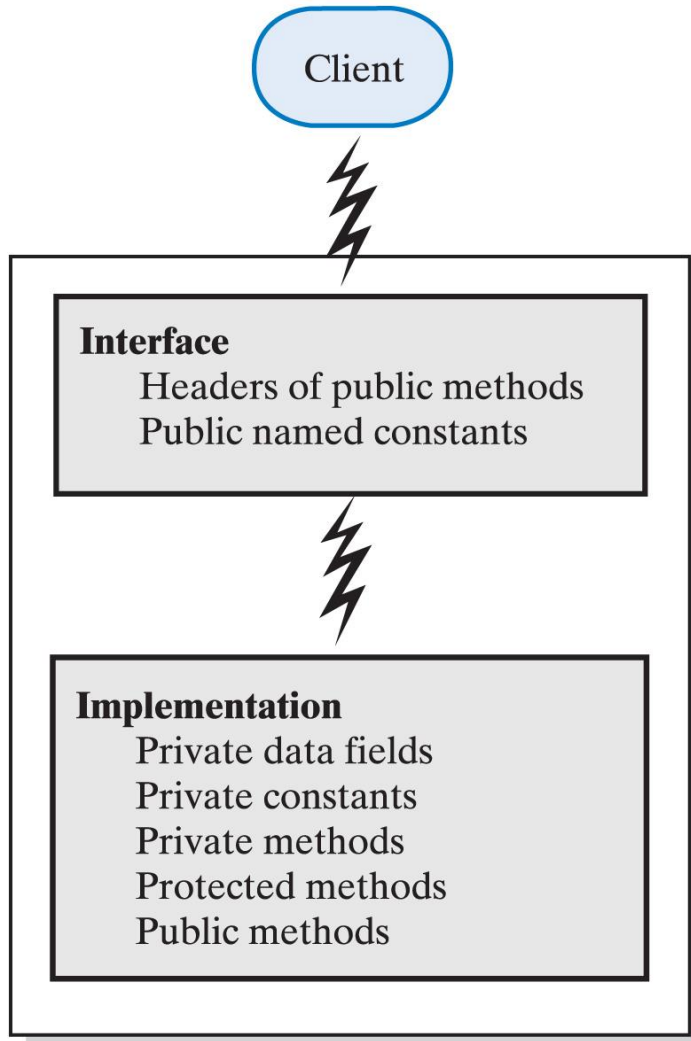
<b>Description</b>	Return the current count value of this object.
<b>Postcondition</b>	This object remains unchanged.
<b>Return</b>	The current value of this counter.



# Implementing Encapsulation

- In object-oriented programming, encapsulation is achieved in a class by
  - ❑ Making its data fields **private**, and
  - ❑ Providing **public methods** for **controlled access** to and manipulation of the data fields

# Data Abstraction



An interface provides well-regulated communication between a hidden implementation and a client.

Carrano (2011)

# To implement an ADT in Java

1. Translate the ADT specification into a Java **interface**
2. Write a **class** which implements the Java interface

# Sample Code: Counter ADT

In Chapter2\carpark\

- **CounterInterface.java**
- **Counter.java**
- **CarParkSystem.java**
  - A GUI application which simulates the use of the counter in a car park with 2 wings

# Summary: Designing & Implementing a new ADT

## Step 1 Write the ADT specification

- Write an ADT specification which describes the characteristics of that data type and the set of operations for manipulating the data. **Should not include any implementation or usage details.**

## Step 2 Implement the ADT

- a. Write a Java interface
  - Include all the operations from the ADT specification
- b. Write a Java class
  - This class implements the Java interface from part a.
  - Determine how to represent the data
  - Implement all the operations from the interface

## Step 3 Use the ADT in a client program or application

# An ADT specification

- Defines the structure, behavior and operations of a “new” data type **without specifying how those structure, behavior and operations are actually implemented.**

# ADT: Rationale

- Separation of Concerns
  - By separating the definition of the data structure from the implementation, we can use the new data structure in programs without regard for its implementation.
  - Hence, the implementation can be changed (improved, updated, etc) and the client programs (i.e. the programs that use the data structure) will only have to change the name of the class that implements the new data structure.

# Post-Lecture Exercise

Past-Year-Question



# PYQ 14-October-2022 Q1a

## BACS2063 DATA STRUCTURES AND ALGORITHMS

### Question 1

If a number is **not** completely **divisible** by two, then it is considered an **odd number**. For such numbers, dividing them by two always leaves a **remainder** of **one**. If a number is completely divisible by two without any remainder, then it is considered an **even number**. Create a simple program that determine if an input number is an odd or even number.

- a) Write an **ADT specification** to represent the above scenario. Include operations that determine odd and even numbers. (10 marks)

# Answer:

## **ADT oddEven**

A number **not** completely **divisible by two**, is an odd number and otherwise is an even number.

Boolean **oddNumber**(Integer number)

**Description:** determine a given value is odd number.

**Pre-condition:** number must be integer and not zero

**Post-condition:** the number remain unchanged

**Returns:** true if odd number; otherwise false

Boolean **evenNumber**(Integer number)

**Description:** determine a given value is even number.

**Pre-condition:** number must be integer and not zero

**Post-condition:** the number remain unchanged.

**Returns:** true if even number; otherwise false

# Learning Outcomes

You should now be able to

- Explain the benefits of abstract data types (ADTs).
- Write ADT specifications.
- Implement ADTs using Java interfaces and classes.



# APPENDIX

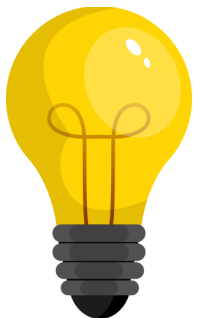
# 1. Perfective (Users Requests)



- Meaning :
  - To cope with **new** requirements due to **users' requests** (eg. new markets, new information required, new technologies)
  - The requests by users may include to improve software **functionality**, **usability** and **performance**.
- Examples
  - Improving **performance** eg. faster **response** to customer service as requested by users.
  - Improving **functionality** eg to add new features or **reports** as requested by users.
  - Improving **usability** eg improving **layouts** of some reports in an inventory system as requested by users.
  - Improving **usability** eg improving the **interface** (to make the system easier to use).

## 2. Corrective (Correction of Errors)

- Meaning
  - To **fix bugs** or faults due to program errors which are unforeseen and arise during operation.
  - To keep the programs working **correctly**.
- Examples
  - Diagnose and fix logic errors or program code.
  - Update software drivers
  - Restore proper configuration settings



### 3. Adaptive (Changes in Environment)

- Meaning
  - To adapt the system due to changing environment :
    - business environment (eg. new regulatory requirements)
    - computer environment (eg. hardware or software changes such as a change of OS)
- Examples
  - Changes to payroll system due to new employment *laws*.
  - Add *new online capability* due to business environment changes
  - Create *new reports* due to business environment changes
  - Add *new data entry field* to input screen due to business environment changes
  - *New systems* will be added eg. OS, hardware, DBMS

