

Recap for Propel Courses

Lesson 00:

People matter, results count.



■ Course Goals

- Recap session for below courses:
 - Programming foundation
 - HTML, Javascript, CSS and XML
 - SQL
 - OOP-UML
 - Software Engineering



■ Course Non Goals

- No complete training session will be conducted for these courses.

Intended Audience

- Course Goals and Non Goals



Copyright © Capgemini 2015. All Rights Reserved 3

Day Wise Schedule

- Day 1
 - Lesson 1: Programming Fundamentals
 - Lesson 2: HTML and Javascript
- Day 2
 - Lesson 3 : XML
 - Lesson 4: DBMS Concepts and SQL
- Day 3
 - Lesson 5 : OOP-UML
 - Lesson 6: Software Engineering



Copyright © Capgemini 2015. All Rights Reserved 4

Table of Contents

- Lesson 1: Programming Fundamentals
 - 1.1 Good programming practices.
 - Readable(Naming Conventions , Comments, Guidelines for writing good code)
 - Maintainable (Remove Hardcoded constants)
 - 1.2 Modular programming
 - 1.3 Coupling and Cohesion
 - 1.4 Composite Datatype
 - 1.5 Robust program
 - 1.6 Review
 - 1.7 Demo
 - 1.8 Case Study



Copyright © Capgemini 2015. All Rights Reserved 5

- Batch Orientation – Day 1 - 1 Hrs
- Lesson 1: Programming Fundamentals– Day 1 - 3 Hrs
 - 1.1 Good programming practices.
 - Readable(Naming Conventions , Comments, Guidelines for writing good code)
 - Maintainable (Remove Hardcoded constants)
 - Modular
 - Coupling and Cohesion
 - Composite Datatype
 - Robust program
 - Pseudocode review checklist
 - 1.2 Demo
 - 1.3 Case Study
- Lesson 2: HTML and Javascript – 4 Hrs
 - 2.1 HTML form element
 - 2.2 HTML 5 new form elements (Number, Date and Email)
 - 2.3 HTML 5 validations
 - 2.2 DOM objects (Document and Form)
 - 2.3 Event handling in javascript
 - 2.4 Demo

Recap for Propel Courses



2.5 Case Study

Table of Contents

- Lesson 2: HTML and Javascript
 - 2.1 HTML form element
 - 2.2 HTML 5 new form elements (Number, Date and Email)
 - 2.3 HTML 5 validations
 - 2.2 DOM objects (Document and Form)
 - 2.3 Event handling in javascript
 - 2.4 Demo
 - 2.5 Case Study



Copyright © Capgemini 2015. All Rights Reserved 6

- Batch Orientation – Day 1 - 1 Hrs
- Lesson 1: Programming Fundamentals – Day 1 - 3 Hrs
 - 1.1 Good programming practices.
 - Readable(Naming Conventions , Comments, Guidelines for writing good code)
 - Maintainable (Remove Hardcoded constants)
 - Modular
 - Coupling and Cohesion
 - Composite Datatype
 - Robust program
 - Pseudocode review checklist
 - 1.2 Demo
 - 1.3 Case Study
- Lesson 2: HTML and Javascript – 4 Hrs
 - 2.1 HTML form element
 - 2.2 HTML 5 new form elements (Number, Date and Email)
 - 2.3 HTML 5 validations
 - 2.2 DOM objects (Document and Form)
 - 2.3 Event handling in javascript
 - 2.4 Demo

Recap for Propel Courses

2.5 Case Study

Table of Contents

- Lesson 3: XML
 - 3.1 Validating xml against xsd
 - 3.2 Simple Type restriction
 - 3.3 Demo
 - 3.4 Case Study
- Lesson 4: DBMS Concepts and SQL
 - 4.1 DDL commands (CREATE, ALTER and DROP)
 - 4.2 Constraints
 - 4.3 Sequence
 - 4.4 DML command (Insert, Update, Delete)
 - 4.5 Select Query, Joins and subquery
 - 4.6 Demos



Copyright © Capgemini 2015. All Rights Reserved 7

➤ **Lesson 5: XML Day 2 – 1.5 Hrs**

- 5.1 Validating xml against xsd
- 5.2 Simple Type restriction
- 5.3 Demo
- 5.4 Case Study

➤ **Lesson 4: DBMS Concepts and SQL – 6.5 Hrs**

- 4.1 DDL commands (CREATE, ALTER and DROP)
- 4.2 Constraints
- 4.3 Sequence
- 4.4 DML command (Insert, Update, Delete)
- 4.5 Select Query, Joins and subquery
- 4.6 Demos

Table of Contents

- Lesson 5: OOP-UML
 - 5.1 Principles in Object-Oriented technology
 - 5.2 UML diagram
 - Use Case Diagram
 - Class Diagram
 - Sequence Diagram
 - 5.3 Demo
 - 5.4 Case study
- Lesson 6: Software Engineering
 - 6.1 Different Phases in Software Engineering
 - Requirements Phase, Design Phase, Construction Phase, Testing and acceptance Phase
 - 6.2 Review and Configuration Management Process
 - 6.3 Case Study



Copyright © Capgemini 2015. All Rights Reserved 8

- **Lesson 5: OOP-UML – 2 Hrs**
 - 5.1 Principles in Object-Oriented technology
 - 5.2 UML diagram
 - Use Case Diagram
 - Class Diagram
 - Sequence Diagram
 - 5.3 Demo
 - 5.4 Case study
- **Lesson 6: Software Engineering – 1 Hrs**
 - 6.1 Different Phases in Software Engineering
 - Requirements Phase, Design Phase, Construction Phase, Testing and acceptance Phase
 - 6.2 Review and Configuration Management Process
 - 6.3 Case Study

Programming Fundamentals

Lesson 01:

Lesson Objectives

- To understand the following concepts
 - Good programming practices.
 - Readable(Naming Conventions , Comments, Guidelines for writing good code)
 - Maintainable (Remove Hardcoded constants)
 - Modular programming
 - Coupling and Cohesion
 - Robust program
 - Composite Datatype
 - Review



Copyright © Capgemini 2015. All Rights Reserved 2

1.1.1 Readable

Naming Conventions

- Use Meaningful names
 - Use Verb-noun format (be specific):
 - Avoid generic names
- Good variable names ensures readability.
- Use naming conventions to distinguish Scope of data.
- Use capitalized names for distinguishing constants among other variables.
- Names should be readable, memorable and appropriate
- A good name tends to express the “what” than the “how”



Copyright © Capgemini 2015. All Rights Reserved 3

Naming Conventions

Meaningful Names:

The name fully and accurately describes what the variable represents

The name should refer to the real-world problem rather than to the programming-language solution

Use Verb-noun format (be specific):

For example: Read-Employee-Record, Calculate-Deductions, Print-Pay-slip

Avoid generic names:

For example: Process-inputs, Handle-calculations

Good variable names are a key element of program readability.

Use naming conventions to distinguish Scope of data like local/global.

For an Example, MAX_USERS_G variable scope is global

Use capitalized names for distinguishing among type names, named constants, enumerated types, and variables.

1.1.1 Readable

Naming Conventions (Contd...)

- Example of Poor Variable Names:

```
X      = X - XX;  
marypoppins = (superman + starship) / god ;  
X      = X + Interest( X1, X );
```

- Example of Good Variable Names:

```
Balance = Balance - LastPayment;  
Balance = Balance + Interest ( CustomerID,Balance);
```



Copyright © Capgemini 2015. All Rights Reserved 4

1.1.1 Readable

Naming Conventions (Contd...)

| Variable Type | Strategy | Variable Name | Example |
|--------------------|----------|---|--|
| Temporary Variable | Bad | Temp | <code>Temp = sqrt(b^2 - 4 *a*c);</code> |
| | Good | Discriminent | <code>Discriminent== sqrt(b^2 - 4 *a*c);</code> |
| Boolean Variable | Bad | NotFound,NotSuccess | <code>If (Found)</code> <code>PRINT "ELEMENT IS PRESENT"</code> |
| | Good | | <code>else</code> <code>PRINT "ELEMENT IS NOT PRESENT"</code> |
| Status Variable | Bad | <code>StatusFlag = 0x80.</code> <code>DataReady=r</code> <code>CharacterType = CONTROL_CHARACTER</code> | |
| | Good | | |
| | | <code>DataReady = TRUE;</code> | |



Copyright © Capgemini 2015. All Rights Reserved 5

Naming Status Variable: Think of a better name than “flag” for status variables.

```

if ( DataReady ) ....
  if ( CharacterType & PRINTABLE_CHAR) ...
    if ( ReportType == AnnualRpt ) ...
      DataReady = TRUE;
      CharacterType = CONTROL_CHARACTER ;
      ReportType = AnnualRpt;
      RecalcNeeded = FALSE;
    
```

CharacterType = CONTROL_CHARACTER is more meaningful than StatusFlag = 0x80.

Naming Temporary Variable: Use meaningful, descriptive names for Temporary variables. Don't use Temp, x or some other vague name.

Bad Temporary variable Name

`Temp = sqrt(b^2 - 4 *a*c);`

A Better approach is

`Discriminent = sqrt(b^2 - 4 *a*c);`

Never use a numeric suffix to differentiate variables

1.1.1 Readable

Optimum Name Length

- The name is long enough that you don't have to puzzle it out.

| | |
|------------|--|
| Too Long | NumberOfPeopleOnTheUSOlympicTeam, NumberOfSeatInTheSaddleDome |
| Too Short | N, NP, NTM, M, MP, Max, Points |
| Just Right | NumTeamMembers, TeamMbrCount, MaxTeamPoints, RecordPoints, cPoints |



Copyright © Capgemini 2015. All Rights Reserved 6

Naming Boolean Variables: Use names that imply true or false like done, error, found, success as boolean variables

Avoid using negative names like NotFound, NotDone and NotSuccessful

“If not notFound” is difficult to read

Better way is to use “if found” instead

Give meaningful names for Loop index.

```
i = 0
ACCEPT Emp, Basic
G = B * 1.8 + 1700
PF = 0.12*B
T = ((G*12 - 150000)*0.3 + 19000)/12
N = G - PF - T - 200
PRINT Emp, B, G, PF, T, N
i=i +1
IF i==10 THEN Stop
Continue
```

In this example variable ‘i’ is used to count how many pay slips have been generated.

‘Rec_Count’ can be used instead of ‘i’

1.1.1 Readable

Kinds of Comments

- Repeat of the code
- Explanation of the code
- Marker in the code `/** @@to do */`
- Summary of the code
- Description of the code's intent



Copyright © Capgemini 2015. All Rights Reserved 7

Kinds of comments

Repeat of the code:

A repetitious comment restates what the code does in different words. It merely gives the reader of the code more to read without providing additional information

Explanatory Comments:

Explanatory comments are typically used to explain complicated, tricky or sensitive pieces of code. If the code is so complicated that it needs to be explained, its nearly always better to improve the code than it is to add comments. Make the code itself cleared and then use summary comments

Marker in the code `/** @@todo */` :

todo is a form of comment used to convey that the code is yet to be completed by replacing todo comment with the functionality logic. For an example, `/**@@todo*/`

Summary of the code : Use comment to provide description of the program like header block comment. For an example,

`//Program Description:`

Description of the code's intent : Use this comment, to describe the layout used

1.1.1 Readable

Commenting Techniques

- Endline comments:
 - Avoid Endline comments on single lines or multiple lines.
 - Use Endline comments to annotate data declarations and to mark ends of blocks.
- Paragraph comments:
 - Focus paragraph comments on the why rather than the how.
 - Use comments to prepare the reader for what is to follow.
 - Avoid abbreviations, comments should be unambiguous.

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 6

Commenting Techniques

Commenting is amenable to several different techniques depending on the level to which the comments apply : program, file, routine, paragraph, or individual line

Endline Comments

Endline comments are comments that appear at the ends of lines of code

Use endline comments to annotate data declarations like

Declare index as integer and store 0. //upper index of an array

Use endline comments to mark ends of blocks

Paragraph Comments

Most comments in a well documented program are one sentence or two sentence comments that describe paragraphs of code

Use to describe the purpose of the block of code

For an example,

```
*****
*      Search for an employee
*****/
```

1.2 Guidelines for writing good code

Guidelines for writing good code

- Program for people, not the machine
- Analyze the case study well
- Design first then code
- Develop in small steps
- Keep your code simple
- Understand the standards
- Document your code
- Paragraph your code
- Paragraph and punctuate multi-line statements
- Use white space
- Specify the order of operations. (Use parenthesis)



Copyright © Capgemini 2015. All Rights Reserved 9

Guidelines for writing good code

- While writing program, keep in your mind that program will be used by people, so make program to be user friendly.
- Before starts with development , analyze the case study well to incorporate all the requirements.
- Do the high level (like database design) and low level designing(pseudocode) of an application before working in coding phase.
- Create program in an incremental approach, so that after implementation of each logic it can be easily tested for finding defects.(As finding defects in earlier stage decreases cost and save development time).
- Make your code to more simple by avoiding the usage of complex data structure/constructs
- Understand the standards:
 - All the coding standards as well as processes should be understandable and apply the same in your code.
 - Believe in them
 - Make them part of your quality assurance process
 - Adopt the standards that make the most sense for you
- Document your code using comments for making it to be more readable
- Paragraph your code by applying modularity to make code reusable.
- Use whitespace as a layout technique to make code more readable
- Specify the order of operations using parenthesis for prioritizing

1.1.2 Maintainable

Maintainable

- If the program is easy to understand and if it is easy to modify then the program is called as maintainable.
- Selection of proper data management technique helps to make code more simpler and maintainable.
- Achieve maintainability by eliminating hard coded constants from the code



Copyright © Capgemini 2015. All Rights Reserved 10

If the program is easy to understand and if it is easy to modify then the program is called as maintainable. Selection of proper data management technique helps to make code more simpler and maintainable. Achieve maintainability by eliminating hard coded constants from the code.

1.1.2 Maintainable

Maintainable - Example

- Program to find the circumference of a circle.

```
BEGIN  
    ACCEPT radius  
    circumference = 2 * 3.14159 * radius  
    PRINT "Circumference of a circle : ", circumference  
END
```

- Better Version

```
BEGIN  
    DECLARE CONSTANT PI AS INTEGER AND STORE 3.14159  
    ACCEPT radius  
    circumference = 2 * PI* radius  
    PRINT "Circumference of a circle : ", circumference  
END
```



Copyright © Capgemini 2015. All Rights Reserved 11

Example:

The given program is used to find the circumference of a circle based on radius. In the given code, hardcoded constant exists which is eliminated by using hard coded constant.

1.1.2 Maintainable

Program for Printing Pay-slip – Example 2

- What does the following Program (example) do?

Version 1

```
BEGIN
ACCEPT ecode, ename, B
G = B * 0.8 + 1700
PF = 0.12*B
T = ((G*12 - 150000)*0.3 + 19000)/12
N = G - PF - T - 200
PRINT ecode, ename, B, G, PF, T, N
END
```



Copyright © Capgemini 2015. All Rights Reserved 12

What are problems in the code above:

Variable names are not meaningful. Understanding meaning of variables G, B, T, N etc is difficult. Hence the code is not easily understandable

We don't understand what the given code is doing?

1.1.2 Maintainable

Program for Printing Pay-slip - Example 2(Contd...)

- Is Version 2 better than version 1 – why?

Version 2

```
BEGIN
ACCEPT ecode, ename, Basic
Gross = Basic * 0.8 + 1700
PF = 0.12 * Basic
Tax = ((Gross * 12 - 150000) *0.3 + 19000)/12
Net = Gross - PF - Tax - 200
PRINT ecode, ename Basic, Gross, PF, Tax, Net
END
```

- What the code is doing is understandable
- The variable names given are meaningful than given in previous example.



Copyright © Capgemini 2015. All Rights Reserved 13

See the above example what it is doing?

It reads employee code, employee name and basic salary from keyboard.
Calculates Gross pay, Provident Fund (PF), Tax and Net pay.
Prints the pay slip

It is understandable because the variable names given are meaningful than given in previous example.

It shows that if you give meaningful variable names then it helps you to understand program better.

1.1.2 Maintainable

Program for Printing Pay-slip - Issues

- What are the issues in understanding the program calculating the gross pay?
 - Poor readability
 - Comments are not added in the code
 - Poor variable names
 - Maintainability
 - Hard-coded constants
 - The program is not modular



Copyright © Capgemini 2015. All Rights Reserved 14

Programming standards that are to be followed are given below:

Use meaningful variable names. It helps to easily understand the program. Avoid using hard-coded constants in the program. Instead, declare them as constants at the beginning of the program, and use these constant names through the program. If value of constant changes later, the value can be modified for the declared constant at one place. The effect will be visible everywhere this constant name is used.

For example: tax rate

Include comments at the header and module level

Don't use literal values in the program instead create constant variable to store fixed literal value for making program to be more maintainable.

Ensure that you have created more modular program.

1.1.2 Maintainable

Program for Printing Pay-slip - Solution

- What solutions do you recommend for these issues?

- Use Header block for comments.
- Use meaningful variable names.
- Eliminate hard coded constants from the code.
- Avoid use of obscure code

For an example:

```
HRA = 0.5 * Basic /* avoid obscure code G = B * 0.8 +  
1700 */
```

```
OPA = 0.3 * Basic /* Offshore project allowance */  
Conveyance = 1700
```

- Use comments to describe program flow or complex sections of code.



Copyright © Capgemini 2015. All Rights Reserved 15

If we use above solution to update the code then the code will become more readable, maintainable.

By reading the code, $G = B * 0.8 + 1700$, we do not understand what is meant by 1700, nor what is meant by 1.8 in the gross calculation.

However, the given code explains that Conveyance is 1700. HRA is 50% of Basic. Offshore Project allowance is 30% of Basic. So do not try to club these equations. Otherwise, maintenance of the code will be difficult, and understandability of code will reduce, as well.

Hence avoid such obscure code. Keep it simplified.

1.1.2 Maintainable

Program for Printing Pay-slip - Solution

- Header Block

```
*****
* File : Example.txt
* Author Name : Capgemini
* Description : Program to Print Pay Slips for all employees
* Version : 3.0
* Last Modified Date : 21-Feb-2015
* Change Description : Added meaningful variable names, made use of
blank lines
*****/
```



Copyright © Capgemini 2015. All Rights Reserved 16

See the given header block of comments

It includes:

File : Give the name of the file.

Author name : Provide the author name who involved in the development of program

Description : Detailed description of the program

Version : Version of the program

Last modified date : Date on which the program is last modified

Change Description: History of changes happened in the program

1.1.2 Maintainable

Program for Printing Pay-slip - Example

- Is Version 3 better than version 1 and 2 – why?

BEGIN

```

ACCEPT ecode, ename, Basic
HRA = 0.5 * Basic    /*** avoid obscure code G = B * 0.8 + 1700 ***/
OPA = 0.3 * Basic    /*** Offshore project allowance ***/
Conveyance = 1700
Gross = Basic + HRA + OPA + Conveyance
Income_Tax = ((Gross * 12 - 150000) * 0.3 + 19000)/12
Provident_Fund = 0.12 * Basic
Prof_Tax = 200
Net = Gross - Provident_Fund - Tax - Prof_Tax
PRINT ecode, ename, Basic, HRA, OPA, Conveyance, Gross
PRINT Provident_Fund, Income_Tax, Prof_Tax, Net

```

END

Obscure code is removed



Copyright © Capgemini 2015. All Rights Reserved 17

Why version 3 is better than version 1 and 2

- Hard coded constants are given proper names. E.g – HRA, OPA, Conveyance.
- The given code is easy to maintain because if conveyance amount , percentage for HRA or OPA changes then we can easily understand where to make the change in the code, which was difficult in version 1 and 2
- Code is readable as naming conventions are followed and layout is applied

Following improvements are required in the code:

- If your code includes any complicated calculations It is necessary to document it in the code. In the above example the calculation of income tax includes many operations. What is meaning of it need to be documented in the code.

Steps involved in income tax calculations

- Calculate annual salary : Gross * 12
- Calculate taxable amount: 1,50,000 will be subtracted from annual salary
- Calculate annual tax: Annual Tax = 30% of taxable amount + 19000
- Calculate monthly tax Monthly tax = Annual tax/12

- The given code is not modular. It doesn't have any modular structure.

If the code is huge. It is performing various functions then it is better to create separate module for each function which increases reusability of the program.

If any of these modules can be reused in some other application Programmer's efforts and time will be saved.

A good example is login module. Almost every application requires login screen which authenticate users. Such modules can be written once and used in multiple applications.

Modular Programming

- A small unit of code for a single purpose
- Intended to operate in a larger program unit
- Can be a function, a method, a procedure or a sub-program or a component
- Is a self contained piece of code , but cannot be independent by itself



Modular : A small unit of code for a single purpose intended to operate in a larger program unit . It can be a function, a method, a procedure or a sub-program or a component. Module is a self contained piece of code , but cannot be independent by itself

Reasons for creating a module

- Reduce complexity
- Better documentation
- Avoid duplication of code
- Avoid dependencies
- Improve performance



Reasons for creating a module

Reduce complexity : By using the abstracting power of modules, complex code can be made to appear simpler and easy to understand

Better documentation : By putting a set code into a well defined module, makes the code self –explanatory

Avoid duplication of code

Avoid dependencies : Sections of code that depend on each other makes changes difficult to incorporate

Improve performance : Easy to test and debug units of code, than a long one

Advantages of Modularity

- Easy to test and debug each unit independently
- Divide work among multiple developers
- Reuse code
- Easy to incorporate changes, as required
- Easy to understand
- Cleaner Code



Advantages of Modularity

Easy to test and debug each unit independently so that defects will be identified at the earlier stage.

Divide work among multiple developers so that application development time will be reduced.

Reuse code: Once a module is written, the same module can be reused in another application.

Easy to incorporate changes, as required so that the code will be more maintainable.

Easy to understand as the code is written in a single unit called module.

Cleaner Code

Characteristics of well defined modules

- They always return same set of results for same set of inputs
- They perform a single well defined functionality
- High cohesion
- Low coupling
- Modular structure
- Meaningful names



Characteristics of well defined modules

They always return same set of results for same set of inputs

They perform a single well defined functionality

High Cohesion – do one thing, and do it well

Low Coupling – reduce dependencies between modules

Ideally when there is “high cohesion” and “low coupling”, one can change the implementation of a routine without impacting the call interface.

For example: A sort routine can change its algorithm from “Bubble” to “Quick sort” – without causing the calling code to break.

Meaningful names

Use Verb-noun format (be specific):

For example: Read-Employee-Record, Calculate-Deductions, Print-Pay-slip

Avoid generic names:

For example: Process-inputs, Handle-calculations

Best practices to follow when creating modules

- Few of the best practices to follow when creating modules
 - Informative module name
 - Module logic should be specific
 - Test each module immediately once it is created
 - Parameter Passing should be accurate.
 - Ensure that there is no "Type mismatch" for any parameter.
 - Ensure that there is no "NOPS" module definition



Copyright © Capgemini 2015. All Rights Reserved 22

Best Practices to follow when creating modules

Informative Module Name: Give the module an informative name like `getProductPrice`, `calculateSalary`, `printDetails`.

Module logic should be specific: Identify a clear purpose for the module before you start writing

Test each module as it is created by performing unit testing

Parameter passing should be accurate: Ensure that the number of parameters, and the sequence is correct for the module call.

Ensure that there is no "Type mismatch" for any parameter.

Ensure that there is no "NOPS"(No Operation) Module definition. i.e. Empty module definition shouldn't be there.

`calculateTotal(Integer price, Integer quantity)`

Refer the valid and invalid statements to invoke a module

`calculateTotal(3,5); //Valid`

`calculateTotal(4,3,4); //Invalid`

`calculateTotal('Test',3); //Invalid`

Example - 1

- Pseudocode to calculate the net billing amount to be paid by the customer. The discount is calculated on Purchase amount as given below
 - 30 % above 5000
 - 20 % for 3001 – 5000
 - 10 % for 1001 – 3000



Solution - 1

- Pseudocode for calculating bill amount. (Not Modularized)

```
BEGIN
    DECLARE PurchaseAmount, DiscountAmount, BillAmount AS INTEGER
    DECLARE TaxPerc AS REAL AND STORE .15
    PROMPT "Enter Purchase amount" AND STORE IN PurchaseAmount
    IF PurchaseAmount < 0 THEN
        DISPLAY "Invalid Amount"
    ELSE IF PurchaseAmount > 5000 THEN
        DiscountAmount = .30 * PurchaseAmount
    ELSE IF PurchaseAmount > 3000 THEN
        DiscountAmount = .20 * PurchaseAmount
    ELSE IF PurchaseAmount > 1000 THEN
        DiscountAmount = .10 * PurchaseAmount
    END IF
    CALCULATE BillAmount = (PurchaseAmount - DiscountAmount) +
                           TaxPerc * (PurchaseAmount - DiscountAmount)
    DISPLAY BillAmount
END
```



Copyright © Capgemini 2015. All Rights Reserved 24

The above pseudocode for calculating bill amount is not modularized

Solution - 2

- Modularized Pseudocode for calculating bill amount.(Contd..)

```
SUB CalculateDiscount(PurchaseAmount)
    DECLARE DiscountAmt AS INTEGER AND STORE 0
    IF PurchaseAmount > 5000 THEN
        DiscountAmt = .30 * PurchaseAmount
    ELSE IF PurchaseAmount > 3000 THEN
        DiscountAmount = .20 * PurchaseAmount
    ELSE IF PurchaseAmount > 1000 THEN
        DiscountAmt = .10 * PurchaseAmount
    END IF
    RETURN DiscountAmt
END SUB
```



Copyright © Capgemini 2015. All Rights Reserved 25

The above pseudocode for calculating bill amount is modularized

Code Considerations During Modularization

- During modularization of the code we need to decide:
 - Input Parameters:
 - For each lower level routine, what input parameters should be passed?
 - Output Parameters:
 - Should the output be in the form of a “parameter” or a “return value”?



Code Considerations

- Analyze parameters and return values for Accept_Employee_Details, Compute_Deductions , Compute_Gross_Pay

```
SUB Accept_Employee_Details()  
    ACCEPT emp_code, Basic  
END SUB
```

```
SUB Compute_Deductions(Basic)  
    Provident_Fund = 0.12 * Basic  
    Prof_Tax = 200  
    Compute_Income_Tax (Basic)  
END SUB
```

```
SUB Compute_Gross_Pay(Basic)  
    HRA = 0.5 * Basic /** House Rent Allowance **/  
    OPA = 0.3 * Basic /** Offshore project allowance **/  
    Conveyance = 1700  
    Gross = Basic + HRA + OPA + Conveyance  
END SUB
```



Guidelines to follow while using arguments

- Identify input and output parameters
- Only include the parameters which are used by the module
- If parameters are related to each other, then pass record as an argument instead of multiple parameters



Guidelines to follow while using arguments

Identify input and output parameters: After analyzing the requirement, identify input and output parameters before writing module code. For an Example, if you want to create a module for implementing a logic related to retrieving product details based on product id, then input parameter is productid and output parameter is variable of type record which contains product details.

Only include the parameters which are used by the module, never pass unused parameters.

If parameters are related to each other, then pass record as an argument instead of multiple parameters which strive for high cohesion and low coupling. For an example, refer the below module code snippet to add an employee details

```
SUB addEmployee(empId, name, salary)
```

```
END SUB
```

Instead use the below module signature

```
SUB addEmployee(Employee emp) //Consider Employee is a  
record
```

```
END SUB
```

Best practice to follow for return values

- Have a single exit point from each module
- Return null values instead of zero length arrays
- Use well defined exceptions and error codes
- Based on the number of values to be returned, use specific return type like array or records.
- Consider the side effects of return values while integrating all the modules together.



Best Practice to follow for return values

Use only one return statement in each module as shown below:

```
SUB checkDigit(number)
```

```
    DELCARE result AS BOOLEAN
```

```
    IF(num>0) THEN
```

```
        result=true
```

```
    ELSE
```

```
        result=false
```

```
    END IF
```

```
    RETURN result
```

```
END SUB
```

Return null values instead of zero length arrays.

Return exceptions/error code if an invalid input is accepted.

```
SUB getProductPrice(productId)
```

```
    DECLARE errorcode AS INTEGER AND STORE 0
```

```
    IF(elementfound(productId)) THEN
```

```
        RETURN productPrice
```

```
    ELSE
```

```
        errorcode = -1
```

```
        RETURN errorcode;
```

```
    END IF
```

```
END SUB
```

Use array as return type if more than one value has to be returned of same type

Programming Fundamentals

|

or use record if more than one value has to be returned of different type.

Coupling

- Coupling or Dependency is the degree to which each program module relies on each other.
- Tightly coupled systems disadvantages:
 - A change in one module forces a ripple-effect of changes in other modules.
 - Assembly of modules might require more effort and/or time due to the increased inter-module dependency.
 - A particular module might be harder to reuse and/or test because dependent modules must be included



If a module does only one thing, we need to pass less parameters.

If a module does too many things, we need to pass more parameters.

Passing more parameters means high coupling.

We should strive for low coupling.

Coupling

- Loosely coupled systems advantages :
 - A change in one module usually does not force a ripple-effect of changes in other modules.
 - Assembly of modules might require less effort and/or time due to the decreased inter-module dependency.
 - A particular module might be easier to reuse and/or test because dependent modules do not need to be included



Low Coupling – reduce dependencies between modules

Note that changes in one routine should not normally impact other routines, as long as the interface is the same.

Remember that, in case too many things are done in one routine, a lot of data needs to be shared. This increases the dependencies, and also the chances of defects

Consider “Smaller interface” (low coupling) versus “long list of parameters” (high)

Consider data sharing through “parameters” (low) versus “global data” or “global files” (high)

Note that passing “flags” that control the processing implies High coupling.

Cohesion

- A cohesion is a measure of how the activities within a single module are related to one another.
- Principle of Cohesion:
 - A module should do one thing and do it well
- If a module follows the given principle, then it is high cohesion.



Writing function with high cohesion is always good practice.

Ideally, we should strive for High Cohesion and Low Coupling.

For example:

Sin (x); Cos (x); Tan (x); (High Cohesion, Low Coupling)

Note:

A good program requires high cohesion and low coupling.

Trig (Type, x)

where type is Sin, Cos, or Tan; (Less Cohesion, High Coupling)

Note:

Since in function trig we are trying to add all three functions together, we require to pass one extra parameter i.e. Type. This parameter indicates whether to calculate Sin, Cos, or Tan.

More number of parameters are required to pass, so it is High Coupling.

b. The function is not performing just a single task, so it is Low Cohesion.

Example

- Example 2: Now, consider the following piece of code as an example
- Review the code for any issues (Coupling, cohesion)

```
SUB ReadCust (filename, custrec)
    custfile=Fopen (filename)
    Fread (custrec, custfile);
END SUB

SUB writeCust (custrec)
    Rewind (custfile);
    Fwrite (custrec, custfile);
    Fclose (custfile);
END SUB

SUB UpdateCust (filename, newbalance)
    ReadCust (filename, custrec);
    Custrec.Balance = newbalance;
    WriteCust (custrec);
END SUB
```



Consider Fopen, Fwrite, Rewind, Fclose are predefined functions.

Fopen : To open a file

Fwrite : To write data to a file

Rewind : To close the opened file

UpdateCust :Moving cursor back to the first position of a file.

Change Request - Example

- Suppose there is a Change Request to code in the above example:
 - Do not update balance, in case the new balance is less than 0. How will you implement this change?
 - Are any problems created due to the change?



Change Request - Example

- Sometimes the change is made as follows

```
SUB UpdateCust (filename, newbalance)
    ReadCust (filename, custrec);
    IF(newbalance >= 0) THEN
        Custrec.Balance = newbalance;
        WriteCust (custrec);
    END IF
END SUB
```

- This means file will not be closed whenever “newbalance < 0”. This is because file gets closed in writecust, and writecust will not be called.
- After a few hours the program will crash saying “too many open files”.



1.3 Coupling and cohesion

Drawbacks in the given code

- ReadCust is doing more than just reading. It is also opening the file.
- WriteCust is doing more than just writing. It is also closing the file.
 - This violates the principle of Cohesion:
- What is the other drawback in the given code with respect to performance overheads?
 - Every time we need to write a record, we are opening and closing the file. This will slow down the program!



Copyright © Capgemini 2015. All Rights Reserved 36

Cohesion:

It means, the function should do one thing only, and the code should not be mixed up.

As a result, the code becomes more readable and more maintainable.

How can we avoid this?

- Use a STATIC variable to represent the STATE of the file.
- Use global variable for accessing the STATE of the file in an application(Not recommended)
- Use higher-level calling routine.
- Encapsulate functions to perform I/O operations in a separate module.



Steps to be taken care for avoiding drawbacks on the implementation of cohesion and coupling:

Use a STATIC variable(cust-file-already-open) to represent the STATE of the file as OPEN or not

We cannot access the STATIC variable cust-file-already-open outside this routine

We will be able to access it outside the routine if we declare it as GLOBAL. However, that is not a good practice.

It is better to open and close the cust-file in the higher-level calling routine. The routine should call UpdateCust only to write the record.

The calls to fread / fwrite / fopen / fclose are encapsulated or wrapped in the modules ReadCust / WriteCust / OpenCust / CloseCust respectively.

Revised Code

```
SUB ReadCust (filename, custrec)
    Fread (custrec, sizeof (custrec), 1, custfile);
END SUB

SUB WriteCust (custrec)
    Fwrite (custrec, sizeof (custrec), 1, custfile);
END SUB

SUB OpenCust (filename, mode, Cfile)
    Cfile = Fopen (filename, mode);
END SUB

SUB CloseCust (Cfile)
    Fclose (Cfile);
END SUB
```



The calls to fread / fwrite / fopen / fclose are encapsulated or wrapped in the modules ReadCust / WriteCust / OpenCust / CloseCust respectively.

Revised Code (Contd..)

```
SUB UpdateCust (filename, newbalance)
    STATIC BOOLEAN cust-file-already-open = FALSE;
    IF (newbalance < 0) THEN
        {return;}
    ELSE IF (NOT cust-file-already-open) THEN
        OpenCust (filename, "r+", Cfile);
    END IF
    ReadCust (Cfile, custrec);
    Custrec.Balance = newbalance;
    WriteCust (custrec);
END SUB
***** CloseCust (Cfile);
Now we need to close the file only once at the end ***/
```



Used STATIC variable(cust-file-already-open) to represent the STATE of the file as OPEN or not

Advantages

- Wrapper modules help isolate system specific code in one place rather than all over the application.
 - They are easier to change during migration to different versions of Compiler or OS.
 - They are extremely useful when porting code:
 - To different platforms, or
 - To different database management systems
 - For example: In the code given in Example 2, "Read_cust" is acting as "wrapper module". It is hiding the details about how to read the file.



What is Robust program ?

- Robust program anticipates common and uncommon problems
- To ensure software is well defended, one should write robust program
- Robust program ensures that software handles invalid inputs reasonably preventing abnormal termination
- The program should terminate gracefully and provide appropriate debugging information for the programmer



Robust program ensures that software handles invalid inputs reasonably preventing abnormal termination. Robust program anticipates common and uncommon problems. To ensure software is well defended, one should write robust program. The program should terminate gracefully and provide appropriate debugging information for the programmer

1.4 Robust Program
Example

- Read the following code for compute_Income_Tax()
- Are there any errors in Compute_Income_Tax module?

```
SUB Compute_Income_Tax(Gross)
    Annual_gross = Gross * 12
    Annual_Tax = 0
    *****
    "for gross between 50 to 60K, tax is 10% of gross over 50K, max 1000"
    "for gross between 60 to 150K, tax is 20% of gross over 60K, max 18000"
    "for gross exceeding 150K, tax is 30% of gross over 150K" *****
    IF (Annual_gross > 50000 and < 60000) THEN
        Annual_Tax = (Annual_gross - 50000) * 0.1
    ELSE IF (Gross > 60000 and < 150000) THEN
        Annual_Tax = 1000 + (Annual_gross - 60000) * 0.2
    ELSE
        Annual_Tax = 1000 + 18000 + (Annual_gross - 150000) * 0.3
    END IF
    Income_Tax = Annual_Tax / 12
END SUB
```



Copyright © Capgemini 2015. All Rights Reserved 42

In the above program we are not considering a case where annual_gross is less than 40000. As a result, in case the statements at the bottom of the program are executed, we will get wrong result.

Note: When we use a nested IF, be careful while writing the last ELSE. This is because, the program will be executed for all unhandled conditions, as well.

The above program has one more mistake. We have used Gross instead of Annual_Gross. This error will not be detected by the compiler because we are using Gross as a variable form during calculation of monthly gross.

This will result in wrong output.

Defects Introduced in the Program

- Do you see any new defects introduced in the program?
 - What tax will be calculated in the Compute_Income_Tax module for an income of 40000?
 - Nested IF-THEN-ELSE should take care of all possible conditions. Is the nested clause doing so?
 - Be careful about the last ELSE – it may end up doing more than it should
 - Program has mistakes in variable name references:
Example: Gross instead of Annual_Gross



For resolving all the defects mentioned in the slide, rework on the code.

1.4 Robust Program
Example

- Is the defects resolved in the below given revised code?

```
SUB Compute_Income_Tax(Gross)
    Annual_gross = Gross * 12
    Annual_Tax = 0
    IF(Annual_gross<=0) THEN
        PRINT "Gross salary cannot be negative"
    IF (Annual_gross>0 and Annual_gross <50000) THEN
        Annual_Tax = (Annual_gross - 49000) * 0.05
    ELSE IF (Annual_gross > =50000 and Annual_gross < 60000) THEN
        Annual_Tax = (Annual_gross - 50000) * 0.1
    ELSE IF (Annual_gross >= 60000 and Annual_gross<= 150000) THEN
        Annual_Tax = 1000 + (Annual_gross - 60000) * 0.2
    ELSE
        Annual_Tax = 1000 + 18000 + (Annual_gross - 150000) * 0.3
    END IF
    Income_Tax = Annual_Tax / 12
END
```



Copyright © Capgemini 2015. All Rights Reserved 44

In the above program we have considered a case where annual_gross is less than 40000. We have renamed, Gross as Annual_Gross. The statements in ELSE block will be executed only if Annual_Gross value is greater than 150000.

Annual Gross salary is calculation is taken care based on the below data

```
/*
*for gross less than 50K, tax is 5% of gross over 49K"
*for gross between 50 to 60K, tax is 10% of gross over 50K, max 1000"
*for gross between 60 to 150K, tax is 20% of gross over 60K, max 18000"
*for gross exceeding 150K, tax is 30% of gross over 150K" *****/

```

Make a Program Robust

- Will the program work (or fail gracefully) with unexpected input?
 - Check if it can display error message as “Input cannot be negative ” .
- Do not assume everything will be alright.
- Check for unexpected inputs or conditions.
- Remember GIGO – Garbage In, Garbage Out.
- Provide meaningful error messages



Is the program robust?

We test the program with different expected values, and it works fine as per our expectation.

However, what if somebody provides an unexpected input?

will the program give proper error message and stop gracefully, or
will the program fail and give some garbage output

Hence to make the program robust, add appropriate messages to handle unexpected input as mentioned in the slide.

How can we make the program robust?

Check if it can display error message as “Input cannot be negative”
Check if it can accept extreme values for inputs.

For example: Basic = 1 crore

Check if it can handle Output / Printer related problems.

Check whether:

“Can the Tax calculated be negative?” or
“Can the Net Pay calculated be negative?”

Difference between correctness and robustness

- Correctness means building code which never returns inaccurate result
 - Safety critical applications tend to focus on correctness , failure to achieve a result being regarded as better than inaccurate result.
 - For an Example, Software which controls a Bank machine should focus on correctness because it is better to return no value than an inaccurate value when an error could mean dispensing or recording wrong amounts of money
- Robustness favors the return of any result even inaccurate one
 - Consumer applications typically favor robustness as any result is better than software crashing
 - For an Example, Web browsers should focus on robustness as they often have to handle invalid input.



Copyright © Capgemini 2015. All Rights Reserved 46

First determine whether you want the program to primarily offer robustness or correctness

Examples of Robustness

Consumer applications typically favor robustness as any result is better than software crashing

You may want a word processing program to sometimes display unwanted characters rather than to shut down when it detects them

Web browsers should focus on robustness as they often have to handle invalid input.

Examples of Correctness

Safety critical applications tend to focus on correctness , failure to achieve a result being regarded as better than inaccurate result.

E.g software which controls radiation equipment for patients is best shut down if it receives bad input for a radiation dosage.

Software which controls a Bank machine should focus on correctness because it is better to return no value than an inaccurate value when an error could mean dispensing or recording wrong amounts of money

What is a record ?

- Record is one of the composite data type, consisting of two or more values or variables stored in consecutive memory positions of different data types.
 - Use them to simplify operations on blocks of data
 - Use them to simplify module parameter lists
- Example:
- Call Hardway (Name, Address, Phone, SSN, Sex, Salary)
 - Call Easyway (EmployeeRec)
 - Use them to reduce maintenance of related data as changes to a record is easier to implement.
 - Used to create user defined data types



Record: Record is one of the composite data type, consisting of two or more values or variables stored in consecutive memory positions .

Example for record:

```
RECORD Employee
    DECLARE ecode AS INTEGER
    DECLARE ename AS STRING
    DECLARE esal AS INTEGER
    DECLARE edept AS STRING
END RECORD
```

The above record is used to hold details about an employee such as employee code, employee name, employee salary and department in which employee is working.

1.5 Composite Datatype

User Defined Data Type - Example

```
RECORD Student
    DECLARE RollNo AS INTEGER
    DECLARE Sname AS STRING
    DECLARE Course AS STRING
END RECORD

//Pseudo Code to read Student data and print it

BEGIN
    ACCEPT Id, Name, Crs

    Student.RollNo=Id
    Student.Sname=Name
    Student.Course=Crs

    PRINT "Student INFO:"
    PRINT "Student Roll Number      : ", Student.RollNo
    PRINT "Student Name       : ", Student.Sname
    PRINT "Student Course     : ", Student.Course
END
```



Copyright © Capgemini 2015. All Rights Reserved 48

The code given in slide is used to maintain information about a student like rollno, sname and course details.

Functionalities implemented in the above code are

- Accepting student details like id, name and course
- Printing the same.

1.6: Review

Code Review

- Static Testing Methods
 - Self Review
 - Done by the author himself with the aid of tools like checklists , review guidelines , rules, etc
 - Code Inspection
 - It is a more systematic and rigorous type of peer review.
 - Code inspection is a set of procedures and error detection techniques for group code reading.
 - Involves reading or visual inspection of a program by a team of people , hence it is a group activity
 - Walk Through
 - Like code inspection it is also an group activity.
 - In Walkthrough meeting, three to five people are involved. Out of the three, one is moderator, the second one is Secretary who is responsible for recording all the errors and the third person plays a role of Test Engineer.



Copyright © Capgemini 2015. All Rights Reserved 49

Review Process

Input : Work Product , Specifications, Checklists, Guidelines, Historical Data

Process

Prepare for Review

Conduct Reviews

Analyze Deviations

Correct Defects

Output : Review Form, reviewed work product

1.6: Review

Pseudocode Review Checklist

- Pseudocode Review Checklist

 Microsoft Excel
17-2003 Worksheet

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 50

Review Process

Input : Work Product , Specifications, Checklists, Guidelines, Historical Data

Process

Prepare for Review

Conduct Reviews

Analyze Deviations

Correct Defects

Output : Review Form, reviewed work product

Demo

- Modular Programming Demo
 - In this demo observe the Naming convention, Layout and comment style followed with modular programming approach.



Case Study

- Rima is working in State Electricity Board Project. She got the following requirement. The following information has to be accepted from the user to calculate the net electricity bill amount.
 - User ID
 - User Name
 - Last month meter reading
 - Current month meter reading



Unit consumed = (Last month meter reading) –
(Current month meter reading)

Net Amount = Unit consumed * 1.15 +Fixed Charge
[Assume Fixed Charge is always Rs.100.]



Copyright © Capgemini 2015. All Rights Reserved 52

Case Study

- Write pseudo code to print the electricity bill. The bill should be in the following format
 - User ID:
 - User Name:
 - Unit Consumed:
 - Net amount:

Take care of good programming practices, modular programming approach and robustness.



Summary

- In this lesson, you have learnt about:
 - Good programming practices.
 - Readable(Naming Conventions , Comments, Guidelines for writing good code)
 - Maintainable (Remove Hardcoded constants)
 - Modular programming
 - Coupling and Cohesion
 - Composite Datatype
 - Robust program
 - Review



Copyright © Capgemini 2015. All Rights Reserved 54

Review Questions

■ Question 1: Why should you make code self documenting ?

- A. To help the review process
- B. To improve the quality of the code
- C. To make it easier to debug
- D. To meet international standards



■ Question 2: How can you minimize hard coding ?

- A. By changing named constants in several places in your code
- B. By using enumerated types instead of named constants
- C. By using named constants to replace Booleans when you require more answers than true or false



Copyright © Capgemini 2015. All Rights Reserved 55

Review Questions

- Question 3: What are the objectives of good layout ?
 - A. An accurate logical structure
 - B. To improve compatibility with compiler
 - C. To improve readability
 - D. To withstand modifications

- Question 4 : What are the guidelines for good layout ?
 - A. Use braces as boundaries to demarcate block of code
 - B. Use indentation to show logical structure of code
 - C. Use line breaks for statements over 80 characters
 - D. Use white space sparingly



Review Questions

- Question 5 : You are writing a module to update the visitor counter on a web page ,what steps can you take to make this module as strong as possible ?
 - A. Call the module pageCountUpdate()
 - B. Call the module webPage6()
 - C. Test the module as soon as it is complete
 - D. Wait until all modules have been added to the program before testing it

- Question 6 : Which of these applications should be robust ?
 - A. Bank machine software
 - B. Radiation machine software
 - C. Web browser software
 - D. Word processing software



Copyright © Capgemini 2015. All Rights Reserved 57

HTML and Javascript

Lesson 02:

Lesson Objectives

- To understand the following concepts
 - HTML form element
 - HTML 5 new form elements (Number, Date and Email)
 - HTML 5 validations
 - DOM objects (Document and Form)
 - Event handling in javascript



HTML Forms for User Input

- HTML forms are used to accept user inputs and then submit data for processing.
- A *form* is an area that contains form elements.
- Types of elements which can be included in a form are
 - Label
 - Single line text field
 - Password field
 - Text area
 - Drop down menu
 - Radio button
 - Checkbox
 - File selector box, etc..



Forms:

HTML forms are used to accept of user input.

A form is an area that contains form elements.

Form elements are elements that allow users to enter information (text fields, text area fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.

Define a form with the `<form>` tag.`</form>`

HTML Forms for User Input

- User input forms are created using <form> tag.
- Syntax:

```
<form method="get/post" action="URL" enctype="Encryption
Type">
    Field definitions
</form>
```

- **action:** the URL of the script
- **method:** the HTTP request method to use, sometimes GET, but usually POST
- **enctype:** Specifies how the data is to be encoded.



Basic syntax for the <form> tag is:

```
<form method="Get or Post" action="URL" enctype="type">
    Field definitions
</form>
```

<form> tag tells a browser that there is a fill-in-the-blank form in this HTML document.

Method:

method attribute states the method to use when you send the form to the server. Two acceptable methods are GET and POST.

GET sends information entered in the form to the server at the end of the URL.

POST sends information entered in the form to the server as a data body/document.

action attribute:

Gives the address of the script that processes the form.

Defines the name of the file to send the content to. File defined here typically does something with the received input.

enctype attribute:

Specifies how the data is to be encoded.

Applies only if you use the POST method. There is only one possible value, the default value “application/x-www-form-urlencoded”.

HTML Forms for User Input

- Some more attributes which can be used in <form> tag are

| Attribute Name | Attribute Value | Description |
|----------------|------------------------------|--|
| name | Form name as a string | Mentions the name of a form. |
| autocomplete | On, off | Specifies whether a form should have autocomplete on or off |
| target | _blank, _self, _parent, _top | Specifies where to display the response that is received after submitting the form |
| novalidate | novalidate | Specifies that the form should not be validated during form submission. |



HTML Form Elements

- <input> element is the most used form tag.
- An <input> tag includes the following attributes
 - name: Name of the field which is required to send data(Key/Value pair) during form submission
 - id: A unique identified of the field
 - value: Sets a default value of the field
 - maxlength: Specifies the maximum number of characters allowed in an <input> element
 - readonly: Specifies that an input field is read-only
 - size: Specifies the width, in characters, of an <input> element
 - Disabled: specified that an input element should be displayed.
 - type attribute of <input> tag specifies the field type



Copyright © Capgemini 2015. All Rights Reserved 6

Input:

The most used form tag is the <input> tag. Input type is specified with the type attribute. Type attributes are:

- Text
- Password
- Hidden
- Radio
- Checkbox
- File
- Button
- Submit/reset

Text-related Elements

- Text related elements can be created as shown below:

| Code | Element |
|-------------------------|----------------------|
| <input type="text"> | Single line text box |
| <input type="password"> | Password field |
| <input type="hidden"> | Hidden field |

- Multiple line text input control
 - If input exceeds more than one line, then create Multi-line input control using HTML <textarea> tag
 - Syntax:
`<textarea rows=" " cols=" " name=" ">`
 - Rows : Number of rows of text area box
 - Cols: Number of columns of text area box
 - Name: name of the element



2.1 HTML form element

Checkbox Element

- If more than one option is required to be selected from multiple options, then create checkbox as shown below:
 - <input type="checkbox">
 - Use checked attribute for selecting any checkbox to be selected by default
- Example:

```
<input type="checkbox" name="hobbies" value="Reading Books"> Reading  
Books <input type="checkbox" name="hobbies" value="Net Surfing"> Net  
Surfing  
<input type="checkbox" name="hobbies" value="Singing" checked> Singing
```

Select your Hobbies: Reading Books Net Surfing Singing



Copyright © Capgemini 2015. All Rights Reserved 6

Radio Button

- If only one option is required to be selected from multiple options, then create radio button as shown below:
 - <input type="radio">
 - Use checked attribute for making a radio button to be selected by default
- Example:

```
<input type="radio" name="sector" value="Public"> Public  
<input type="radio" name="sector" value="Private"> Private
```

Select your sector in which you are working: Public Private



Drop down list

- Drop down list allow the user to select one or more values from a pre-determined options
- Tags for creating drop down list with options are:
 - <select> : Creates drop down list
 - <option>: Defines an option in a select list.

| Tag name | Attribute | Description |
|----------|-----------|---|
| <select> | Name | Defines a name for the drop down list |
| | Size | Defines the number of visible options in a drop down list |
| | Multiple | Allow to select multiple options at once |
| | Disabled | Disable drop down list |
| <option> | value | Specifies the value to be sent to a server |
| | Selected | Makes option to be selected by default |



2.1 HTML form element

Drop down list - Example

```
<!DOCTYPE html>
<html>
<body>Select a country:
    <select name="country">
        <option value="Germany">Germany</option>
        <option value="India" selected>India</option>
        <option value="China">China</option>
        <option value="Japan" >Japan</option>
    </select>
</body>
</html>
```

Select a country: India ▾



Copyright © Capgemini 2015. All Rights Reserved 11

File Upload

- File upload will allow the user to upload a file from the desktop to an application in browser.
- The below code is used to define a file-select field and a "Browse..." button (for file uploads):
 - <input type="file"/>
- In <form> tag, use enctype="multipart/form-data" if file need to be uploaded using file selector form element.

| Attribute | Description |
|-----------|---|
| Name | Defines a name for the file upload dialog box |
| Disabled | Disable element |
| Accept | Specify MIME type to describe the file type which accepts by a server |



2.1 HTML form element

File Upload- Example

```
<!DOCTYPE html>
<html>
<body>
    <form method="post" action="success.html"
          enctype="multipart/formdata">
        Select a photo to upload:
        <input type="file" name="photo"/>
    </form>
</body>
</html>
```

Select a photo to upload:



Copyright © Capgemini 2015. All Rights Reserved 13

Button

- Different types of button which is possible to be created in HTML5 are as shown below:

| Field type | Element | |
|-----------------------|---------------|--|
| <input type="button"> | Button | A clickable button, that activates a JavaScript when it is clicked |
| <input type="submit"> | Submit button | Defines a button for submitting a form |
| <input type="reset"> | Reset button | Define a reset button (resets all form values to default values) |



2.2 : HTML 5 new form elements

Number

- Up and down button provided to increase and decrease the value.
- Min and max parameters provided to limit the values.
- Browser will treat it as simple textfield if it doesn't support this type.
- Syntax is

```
<input id="movie" type="number" value="0"/>
```

```
<input id="user_lic" type="number" min="5" max="30" step="5" value="" />
```

- Limiting the values for this field...

Copyright © Capgemini 2015. All Rights Reserved 15

Number:

Asking for a number is trickier than asking for an email address or web address.

First of all, numbers are more complicated than you might think. You don't often ask for "just a number." It's more likely that you'll ask for a number in a particular range. You may only want certain kinds of numbers within that range- maybe whole numbers but not fractions or decimals.

Example:

```
<input type="number" min="0" max="10" step="2" value="6">
```

Let's take that one attribute at a time.

`type="number"` means that this is a number field.

`min="0"` specifies the minimum acceptable value for this field.

`max="10"` is the maximum acceptable value.

`step="2"`, combined with the `min` value, defines the acceptable numbers in the range: 0, 2, 4, and so on, up to the `maxvalue`.

`value="6"` is the default value.

2.2 : HTML 5 new form elements

Date

- Date
 - Important and mostly used element
 - Simple to implement
 - Before HTML5, programmers used to write lines of javascript code for date picker
 - Input type for date:- date, week, month, time, datetime (gives UTC time), datetime-local (local time)
- Syntax is

```
<input id="meeting" type="date" value="" />
```



The screenshot shows a web browser window with an input field labeled "Meeting Date" containing "2011-01-15". A calendar dropdown is open, showing January 2011. The date "15" is highlighted in blue, indicating it is selected. The calendar grid includes columns for Monday through Sunday. Below the calendar is a "Today" button and the Opera 11 logo.

Meeting Date 2011-01-15

| Mon | Tue | Wed | Thu | Fri | Sat | Sun |
|-----|-----|-----|-----|-----|-----|-----|
| 31 | 30 | 29 | 30 | 31 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 1 | 2 | 3 | 4 | 5 | 6 |

Today Opera 11

Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2011. All Rights Reserved 16

Date:

HTML 4 did not include a date picker control. JavaScript frameworks have picked up

the slack (Dojo, jQuery UI, YUI, Closure Library), but of course each of these solutions requires “buying into” the framework on which the date picker is built.

HTML5 finally defines a way to include a native date picker control without having to

script it yourself. In fact, it defines six input types: date, month, week, time, date +

time, and date + time - timezone.

So far, support is... sparse.

2.2 : HTML 5 new form elements

Email

- Email - This field is used to check whether the string entered by the user is valid email id or not.
- Syntax is –

```
<input id="email" name="email" type="email" />
```
- Browser's that don't support this field will treat this as a simple text field
- This is how it looks like on form

Email:
nettutsplus.com is not
a legal email address

Opera
10.63

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 17

Email:

The email type is used for input fields that should contain an e-mail address.

The

value of the email field is automatically validated when the form is submitted.

Example

E-mail: `<input type="email" name="user_email" />`

Safari on the iPhone recognizes the email input type, and changes the on-screen

keyboard to match it (adds @ and .com options).

2.3 HTML 5 validations

Required

- Required - A field with "required" attribute must be filled in with value before submission of a form
- Syntax is –

```
<input name="name" type="text" required />
```
- The picture below shows us how Firefox and Opera prompt user to fill in value if a "Required field" is left blank upon submission

The screenshot displays two browser interfaces side-by-side. On the left is Firefox 4 Beta, showing a red box shadow around an empty input field and a tooltip 'Please fill out this field.' above it. On the right is Opera 9, showing a red rounded rectangle around the same input field with the text 'This is a required field.' inside. Both browsers have a 'Submit' button next to the input field. Below the screenshots are the Firefox 4 logo and the Opera 9 logo. At the bottom of the slide is the Capgemini logo and copyright information: 'Copyright © Capgemini 2010. All Rights Reserved' and '16'.

Required:

If the required attribute is present, then the field must contain a value when the form

is submitted. This informs the (HTML5-aware) web browser that the field is to be

considered mandatory. Different browsers may mark the input box in some way

(Firefox 4 Beta adds a red box-shadow by default), display a warning (Opera) or

even prevent the form from being submitted if this field has no value.

Hopefully these

behaviours will converge in future releases.

Here's an example of an input field for a required email address that ensures that the

field has a value and that the value is a valid email address.

Example:

```
<input type="email" id="email_addr" name="email_addr" required />
```

2.3 HTML 5 validations

Pattern

- Pattern - A value filled in the field must be checked against the regular expression specified in pattern attribute.
- Syntax is –

```
Pincode: <input type="text" name="pin_code" pattern="[0-9]{6}" title="666666">
```

- The picture below shows us how browser prompt user to fill in valid value if a “field with Pattern attribute” is filled with invalid value upon submission

Pincode: 773 X

Submit You must use this format: 666666

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 19

Pattern:

The pattern attribute specifies a regular expression that the <input> element's value is checked against.

Note: The pattern attribute works with the following input types: text, date, search, url, tel, email, and password.

Tip: Use the global title attribute to describe the pattern to help the user.

Example:

```
<input type="text" name="country_code"
pattern="[A-Za-z]{3}" title="Three letter country code">
<input type="password" name="pw" pattern=".{6,}" title="Six or more
characters">
<input type="password" name="pw" pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-
Z]).{8,}" title="Must contain at least one number and one uppercase and
lowercase letter, and at least 8 or more characters">
<input type="email" name="email" pattern="[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-
z]{2,3}$">
<input type="url" name="website" pattern="https?:\/\/.+\" title="Include http://">
```

Refer <http://www.html5pattern.com> to explore some more patterns.

2.3 : HTML 5 validations

Placeholder

- Place Holder - A placeholder is a textbox that hold a text in lighter shade when there is no value and not focused
- Syntax is –

```
<input id="first_name" placeholder="This is a placeholder">
```

- This is how place holder looks like on supporting browser
 
- Once the textbox gets focus, the text goes off and you shall input your own text
 

 Firefox 3.7  Safari 4.0  Chrome 4.0  Opera 11  Android 2.2

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

Place Holder:

The first improvement HTML5 brings to web forms is the ability to set placeholder

text in an input field. Placeholder text is displayed inside the input field as long as the

field is empty and not focused. As soon as you click on (or tab to) the input field, the

placeholder text disappears.

Here's how you can include placeholder text in your own web forms:

```
<form>
  <input name="name" placeholder="Enter your name">
  <input type="submit" value="Search">
</form>
```

Browser's that don't support placeholder attribute will simply ignore it. But if you

want to make it work in other browsers, you can use some JavaScript to create

The same behavior. There is an excellent jQuery plugin called HTML5 Placeholder

Plugin that will go through all input fields with placeholders attached to them and

make them work in all browsers.

Working With Document Object

- Container for all HTML HEAD and BODY objects associated within tags
- Provides access to page elements from your script
 - This includes form, link, anchor, as well as global Document properties such as background and foreground colors



Document object is part of the Window object. It is used to access all elements in a page. It provides access to the elements in an HTML page from within the script.

This includes the properties of every form, link and anchor (and, where applicable, any sub-elements), as well as global document properties such as background and foreground colors.

2.4 DOM objects (Document and Form)

Document Object Properties

- document.anchors
- document.forms
- document.links
- document.title

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 22

document.anchors - Returns a collection of all `<a>` elements in the document that have a name attribute

document.forms - An `HTMLCollection` Object, representing all `<form>` elements in the document. The elements in the collection are sorted as they appear in the source code.

document.links - Returns a collection of all `<a>` and `<area>` elements in the document that have a href attribute

document.title - Sets or returns the title of the document

| Property | Description |
|---|---|
| alinkColor vlinkColor bgColor fgColor linkColor | Get and set the properties of document – activated link, visited link, background color, foreground color (text) and hyperlink color. |
| anchors[], forms[], links[] | These properties retrieve array of values respectively as present in the document object |
| title | Gets the title of the document which occurs between the <code>TITLE</code> tags. |

2.4 DOM objects (Document and Form)

Document Object Methods

- write(), writeln()
- getElementById()
- getElementsByTagName()
- getElementsByName()
- getElementsByClassName()

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 23

write() - Writes HTML expressions or JavaScript code to a document

writeln() - Same as write(), but adds a newline character after each statement

getElementById() - Returns the element that has the ID attribute with the specified value

getElementsByTagName() - Returns a NodeList containing all elements with the specified tag name

getElementsByName() - Returns a NodeList containing all elements with a specified name

getElementsByClassName() - Returns a NodeList containing all elements with the specified class name

| Property | Description |
|---|--|
| write("string1", ...) writeln("string1", ..) | Both of these methods send text to a document for display in its window. The only difference between the two methods is that <code>document.writeln()</code> appends a carriage return to the end of the string it sends to the document (you must still write a to insert a line break). |
| getElementById("#ara1") | This method locates the element whose id has been passed. The text within this element can then be accessed using properties innerHTML or innerText |
| getElementsByTagName("p") | This method locates all the elements which match the tagname passed. Each element of this type of tag can then be accessed in an array like manner |
| getElementsByName() | This method locates all the elements which match the name passed. Same name to many elements is usually given for radio buttons. |
| getElementsByClassName() | This method locates all the elements which match the class name passed. |

2.4 DOM objects (Document and Form)

Document Object Methods

```
<!DOCTYPE html>
<html>
<style>
.one{color:blue}
</style>
<body>
<p id="p1"> India is great </p>
<p name="para1"> India is secular country </p>
<p class="one"> India is democratic country</p>
<p name="para1"> It is the seventh-largest country by
area</p>
</body>
</html>
```

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 24

write() - Writes HTML expressions or JavaScript code to a document

writeln() - Same as write(), but adds a newline character after each statement

getElementById() - Returns the element that has the ID attribute with the specified value

getElementsByTagName() - Returns a NodeList containing all elements with the specified tag name

getElementsByName() - Returns a NodeList containing all elements with a specified name

getElementsByClassName() - Returns a NodeList containing all elements with the specified class name

| Property | Description |
|---|---|
| write("string1", ...) writeln("string1", ..) | Both of these methods send text to a document for display in its window. The only difference between the two methods is that <code>document.writeln()</code> appends a carriage return to the end of the string it sends to the document (you must still write a <code> </code> to insert a line break). |
| getElementById("#para1") | This method locates the element whose id has been passed. The text within this element can then be accessed using properties <code>innerHTML</code> or <code>innerText</code> |
| getElementsByTagName("p") | This method locates all the elements which match the tagname passed. Each element of this type of tag can then be accessed in an array like manner |
| getElementsByName() | This method locates all the elements which match the name passed. Same name to many elements is usually given for radio buttons. |
| getElementsByClass() | This method locates all the elements which match the class name passed. |

`write()` - Writes HTML expressions or JavaScript code to a document

`writeln()` - Same as `write()`, but adds a newline character after each statement

`getElementById()` - Returns the element that has the ID attribute with the specified value

`getElementsByTagName()` - Returns a NodeList containing all elements with the specified tag name

`getElementsByName()` - Returns a NodeList containing all elements with a specified name

`getElementsByClassName()` - Returns a NodeList containing all elements with the specified class name

| Property | Description |
|---|---|
| <code>write("string1", ...)</code> <code>writeln("string1", ..)</code> | Both of these methods send text to a document for display in its window. The only difference between the two methods is that <code>document.writeln()</code> appends a carriage return to the end of the string it sends to the document (you must still write a <code> </code> to insert a line break). |
| <code>getElementById("#para1")</code> | This method locates the element whose id has been passed. The text within this element can then be accessed using properties <code>innerHTML</code> or <code>innerText</code> |
| <code>getElementsByTagName("p")</code> | This method locates all the elements which match the tagname passed. Each element of this type of tag can then be accessed in an array like manner |
| <code>getElementsByName()</code> | This method locates all the elements which match the name passed. Same name to many elements is usually given for radio buttons. |
| <code>getElementsByClassName()</code> | This method locates all the elements which match the class name passed. |

Handlers Form Object

| Properties | Methods | Event Handlers |
|-------------|----------|----------------|
| action | reset() | onReset |
| elements[] | submit() | onSubmit |
| enctype | | |
| length | | |
| method | | |
| name | | |
| target | | |



Working with Form Objects: Form Object Properties:

A form element provides the only way that users can enter textual information or make a selection from a predetermined set of choices, whether those choices appear in the form of an on/off checkbox, one of a set of mutually exclusive radio buttons, or a selection from a list.

| Property/ Method/ Events | Description |
|--------------------------------|--|
| action | This property is the same as the value you assign to the ACTION attribute of a <FORM> tag. The value is typically a URL on the server where queries or postings are sent for submission. |
| elements[] | Returns an array of elements. It includes all the user interface elements defined for a form: text fields, buttons, radio buttons, checkboxes, selection lists, and more. |
| encoding | You can define a form to alert a server that the data being submitted is in a MIME type. This property reflects the setting of the ENCTYPE attribute in the form definition. The default value is an empty string. |
| method | A form's method property is either the GET or POST values assigned to the METHOD attribute in a <FORM> tag. |

2.4 DOM objects (Document and Form)

Text-Related Objects

- Text

Enter name

Enter Password

Enter Address

- Password
- TextArea
- Hidden Objects

Not visible on browser

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 28

Text-Related Objects:

Text Objects : The text object is the primary medium for capturing user-entered text.

Password Object: A password-style field looks like a text object, but when the user types something into the field, only asterisks or bullets (depending on your operating system) appears in the field.

Textarea Object: A textarea object closely resembles a text object, except for attributes that define its physical appearance on the page.

Hidden object: A hidden object is a simple string holder within a form object whose contents are not visible to the user of your Web page. With no methods or event handlers, the hidden object's value to your scripting is as a delivery vehicle for strings that your scripts need for reference values or other hard-wired data.

Text-Related Objects (Contd..)

| Properties | Methods | Event Handlers |
|--------------|----------|----------------|
| defaultValue | blur() | OnBlur |
| name | focus() | OnChange |
| type | select() | OnFocus |
| value | | |



The properties, methods and event handlers are same for text object, text area and Password. For hidden object the properties are same but no methods and event handlers are associated with this object.

| Property/ Events/ Methods | Description |
|---------------------------------|--|
| defaultValue | Specifies or returns a defaultValue for a text related objects. |
| name | This property can be used to reference the text object in the script. |
| type | Returns the type of text related object |
| value | A reference to an object's value property returns the string currently displayed in the field. |

2.4 DOM objects (Document and Form)

Button Objects

- Button
- Reset
- Submit

| Properties | Methods | Event Handlers |
|------------|---------|----------------|
| name | click() | OnClick |
| type | | |
| value | | |

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 31

Button Objects: Button, Submit and Reset

| Property | Description |
|----------|--|
| name | You may need to retrieve this property in a general-purpose function handler called by multiple buttons in a document. The function can test for a button name and perform the necessary statements for that button. |
| type | The precise value of the type property echoes the setting of the TYPE attribute of the <INPUT> tag that defined the object: button; submit; or reset. |
| value | A button's visible label is determined by the VALUE property. |
| click() | A button's click() method should replicate, via scripting, the human action of clicking that button. |
| onClick | Virtually all button action takes place in response to the onClick event handler. A click is defined as a press and release of the mouse button while the screen pointer rests atop the button. |

Check Box And Radio Objects

- Checkbox
- Radio

| Properties | Methods | Event Handlers |
|----------------|---------|----------------|
| checked | click() | OnClick |
| defaultChecked | | |
| name | | |
| type | | |
| value | | |



Copyright © Capgemini 2015. All Rights Reserved 32

Checkbox object:

| Property/ Events/ Methods | Description |
|---------------------------------|--|
| checked | The simplest property of a checkbox gets or lets you set whether or not a checkbox is checked. The value is true for a checked box and false for an unchecked box. Only one radio button in a group can be highlighted checked) at a time. That one button's checked property is set to true, whereas all others in the group are set to false. |
| defaultChecked | If you add the CHECKED attribute to the <INPUT> definition for a checkbox or radio button, the defaultChecked property for that object is true; otherwise, false. |
| name | The name property allows user to access name for the checkbox or radio button through script. |
| type | Use the type property to help you identify a checkbox object or a radio button object from an unknown group of form elements. |

2.4 DOM objects (Document and Form)

Select Object

| Properties | Methods | Event Handlers |
|---------------|---------|----------------|
| length | blur() | onChange |
| name | focus() | onFocus |
| selectedIndex | | onBlur |
| type | | |

OPTION

Properties

Default Selected

text

selected

Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 34

| Property/ Methods/ Events | Description |
|---------------------------------|---|
| length | Returns the number of items available in the list. A select object with three choices in it has a length property of 3. |
| Name | A select object's name property is the string you assign to the object by way of its NAME attribute in the object's <SELECT> tag which can be |
| selectedIndex | When a user clicks on a choice in a selection list, the selectedIndex property changes to a number corresponding to that item in the list. |
| type | Use the type property to help you identify a select object from an unknown group of form elements. |
| blur() focus() | Your scripts can bring focus to a select object by invoking the object's focus() method. To remove focus from an object, invoke its blur() method. These methods work identically with their counterparts in the text object. |
| onChange | As a user clicks on a new choice in a select object, the object receives a change event that can be captured by the onChange event handler. |

Event Handlers

- Specify how an object reacts to an event
 - Event can be triggered by a user action or a browser action.
- There are two ways to map functions to events
 - Event handlers as methods:

```
document.formName.button1.onclick=f1()
```
 - Event handlers as properties:

```
<INPUT TYPE="button" NAME="button1" onClick="f1()">
```



Object Event Handlers

Event handlers specify how an object reacts to an event, whether the event is triggered by a user action (for example, a button click) or a browser action (for example, the completion of a document load). Event Handlers can be specified as methods or they can be specified using attributes in tags.

2.5 Event handling in javascript

Event Handlers

| Event | Description |
|-------------|--|
| onchange | An HTML element has been changed |
| onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 37

Demo

- Demo on:
 - SampleForm.html (html File)
 - Validation.js (javascript file)



Copyright © Capgemini 2015. All Rights Reserved 38

Case Study

- Create a **prob3.html** page as shown in the below figure. The page should be submitted on clicking the **Submit** button when all the form fields are properly validated.

| | |
|---------------------------------------|--|
| Name: | <input type="text"/> |
| Date of Birth: | <input type="text"/> mm/dd/yyyy |
| Phone Number: | <input type="text"/> |
| Email: | <input type="text"/> |
| Graduation Level: | <input type="radio"/> UG <input type="radio"/> PG |
| Qualification: | <input type="button" value="Select your qualification ▾"/> |
| <input type="button" value="Submit"/> | |



Copyright © Capgemini 2015. All Rights Reserved 39

Case Study

- None of the fields should be empty (Use HTML 5 required attributes)
- Name field should be between 3 to 10 characters (Use HTML 5 pattern attributes)
- For Date of Birth format use HTML 5 date control
- Phone Number should be in xxx-xxxx-xxxx format (Use HTML 5 pattern attributes)
- Email ID should be valid. (Use HTML 5 email control)
- Use placeholder attribute to denote the expected pattern format to the user
- Based on graduation level selected, qualification need to be populated automatically. For an example, if graduation level selected is UG, then qualification should be B.Sc, B.A, B.Com, etc... If graduation level selected is PG, then qualification should be M.A, M.Tech, MCA, MBA, etc... (Call function on onChange event)
- Calculate age of the person and display all the details in a new popup window when "Submit" button is clicked. Details should be printed in the specified format as given below:
 - Name:
 - Age:
 - Phone Number:
 - Email:
 - Graduation Level:
 - Qualification:
- Display the appropriate error message when the validation condition fails.



Copyright © Capgemini 2015. All Rights Reserved 40

Hint: Use below code for age calculation:

```
today = new Date();
dob=document.frm.dob.value;
var age = today.getYear()-dob.getYear();
```

Summary

▪ In this lesson, you have learnt about:

- HTML form element
- HTML 5 new form elements (Number, Date and Email)
- HTML 5 validations
- DOM objects (Document and Form)
- Event handling in javascript



Copyright © Capgemini 2015. All Rights Reserved 41

Review Questions

- Question 1: A _____ is a textbox that hold a text in lighter shade when there is no value and not focused
- Question 2: Radio Buttons are used when you want the user to select:
 - Option 1: one of a limited number of choices.
 - Option 2: one or more options of a limited number of choices.
 - Option 3: many of unlimited number of choices.
- Question 3: METHOD attribute states the method to use when you send the form to the server.
 - True/ False
- Question 4: The _____ attribute of form specifies how the data is to be encoded.



XML

Lesson 03:

Lesson Objectives

- To understand the following concepts

- Validating xml against xsd
- Simple Type restriction



Introduction to XML Schema

- The XML Schema Definition Language is an XML language for describing and constraining the content of XML documents
- XML Schema is a W3C recommendation
- XML Schema defines what it means for an XML document to be valid
- XML Schema are a radical departure from Document Type Definitions (DTDs), the existing schema mechanism inherited from SGML



Copyright © Capgemini 2015. All Rights Reserved 3

What You Should Already Know

Before you continue, you should have a basic understanding of the following:

HTML/ XHTML

XML and XML Namespaces

Introduction to Schema:

An XML document is essentially a structured medium for storing information. In order to assess the validity of a XML document, you need to establish exactly to which structure the information within the document must adhere. This is accomplished with schema.

A schema describes the arrangement of markup and character data within a valid XML document. It describes the grammar, vocabulary, structure, datatypes, etc. of a XML document. A traditional schema solution is DTD. We are already familiar with DTD and XML namespaces.

Why Use XML Schemas?

- XML Schemas
 - support data types
 - use XML syntax
 - secure data communication
 - are extensible
 - Well-Formed is not enough



Copyright © Capgemini 2015. All Rights Reserved 4

Why Use XML Schemas?

As mentioned earlier XML Schemas have advantages over using DTD. Let us now see some more reasons why we should use XML Schemas.

Support Data Types: XML Schemas have support for datatypes. This makes it simple to describe allowable document content, to validate the data correctness. It is also easy to work with data from database, defining restrictions on data and/or data formats. It also allows conversion of data between different data types.

Use of XML Syntax: When writing XML Schemas you follow XML syntax. Hence you can use the XML Editors and Parsers to work with the Schema files. In addition to this you also do not need to learn a different language.

Secure Data Communication: During data transfer it is essential that both dispatcher and receiver of the data have the same understanding about the transferred content. The dispatcher will have to depict the data in such a way that it is understood by the receiver.

Are extensible: XML schemas can be inherited i.e one XML schema can be extended by another XML schema. You can also create your own datatypes from standard datatypes.

3.1 : Validating xml against xsd

XML Schema

- An XML Schema defines:
 - Elements that can appear in a document
 - Attributes that can appear in a document
 - The elements that are child elements
 - The order of child elements
 - The number of child elements
 - The criteria whether an element is empty or can include text
 - Data types for elements and attributes
 - Default and fixed values for elements and attributes

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 5

Advantages of Schemas over DTD:

Example:

Consider the following example:

```
<!ELEMENT pin-code #PCDATA>
```

Now, consider the following statement:

```
<pin-code>ABC-123444-hhh</pin-code>
```

It is both well-formed and valid even though ABC-123444-hhh certainly does not represent a pin code in any form.

The data-type constraints available in schemas can allow the schema designer to limit the content of the pin-code element to a six digits number, for example, 400090.

XML Schemas are the successors of DTDs.

3.1 :Validating xml against xsd

Namespaces

- XML Namespaces provide a method to avoid element name conflicts
- Name Conflicts: In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications
- XML Namespaces provides a method to avoid element name conflicts



Copyright © Capgemini 2015. All Rights Reserved 6

3.1 :Validating xml against xsd

Namespaces

```
<table>
<tr>
<td>Apples</td>
<td>Bananas</td>
</tr>
</table>
```

```
<table>
<name>African Coffee
Table</name>
<width>80</width>
<length>120</length>
</table>
```

```
<tables>
<table> ....</table>
<table> ....</table>
</tables>
```

How do you
differentiate
between these
table?

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 7

Namespaces:

If the XML fragments in the above slide were added together, then there would be a name conflict. Both contain a `<table>` element, but the elements have different content and meaning.

An XML parser will not know how to handle these differences.

3.1 :Validating xml against xsd

Namespaces

- The namespace attribute is placed in the start tag of an element and has the following syntax:

xmlns:namespace-
prefix="namespace"

- The W3C namespace specification states that the namespace itself should be an Uniform Resource Identifier (URI)
- When a namespace is defined in the start tag of an element, all child elements with the same prefix are associated with the same namespace



Copyright © Capgemini 2015. All Rights Reserved 6

Solving the Name Conflict Using a Prefix

- Code Snippet

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/">
<h:tr>
<h:td>Apples</h:td><h:td>Bananas</h:td>
</h:tr>
</h:table>
<f:table xmlns:f="http://www.w3schools.com/furniture">
```



Solving the Name Conflict using a Prefix:

In the example in the above slide, there will be no conflict because the two `<table>` elements have different names. This XML carries information about an HTML table, and a piece of furniture.

When using prefixes in XML, a so-called namespace for the prefix must be defined.

The namespace is defined by the `xmlns` attribute in the start tag of an element.

The namespace declaration has the following syntax.
`xmlns:prefix="URI"`.

In the example on the above slide, the `xmlns` attribute in the `<table>` tag gives the `h:` and `f:` prefixes a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Solving the Name Conflict Using a Prefix

- Code Snippet continued

```
<f:name>African Coffee Table</f:name>
<f:width>80</f:width><f:length>120</f:length>
</f:table>
</root>
```



Illustration(Message.xsd)

- Let us see an example on writing a schema definition:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
<xs:element name="message">
<xs:complexType>
<xs:sequence>
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
```



Creating Schema Document:

The `<schema>` element is the root element of every XML Schema. The `<schema>` element contains some attributes. A schema declaration often looks like something as shown in the above slide.

`xmlns:xs= http://www.w3.org/2001/XMLSchema`

It implies that the elements and data types used in the schema are from "http://www.w3.org/2001/XMLSchema" namespace. It also signifies that any elements and datatypes referred from here should have the prefix "xs".

Some more optional attributes:

`targetNamespace="patniNamespace"`

This value is a unique identifier. The value could be anything. Place this attribute at the top of the XSD means all entities are part of the namespace

`xmlns= " http://www.w3.org/2001/XMLSchema "`

It indicates that the default namespace is "http://www.w3.org/2001/XMLSchema".

`elementFormDefault="qualified"`

It signifies that any elements used by the XML instance document that were declared in this schema must be qualified by namespace.

3.1 :Validating xml against xsd

Illustration(Message.xsd)

- Code Snippet continued

```
<xs:element name="subject" type="xs:string"/>
<xs:element name="text" type="xs:string"/>
<xs:attribute name="priority" type="xs:string"      use="required"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```



Copyright © Capgemini 2015. All Rights Reserved 12

Using XSD in XML Document

- Example:

```
<note xmlns="http://www.w3.org/2001/XMLSchema "
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="message.xsd">
```



Writing a Schema Definition for an XML File:

Using XSD in XML Document:

`xmlns=" http://www.w3.org/2001/XMLSchema"` indicates the default namespace declaration which tells the schema-validator that all the elements used in this XML document are declared in the `"http://www.w3.org/2001/XMLSchema"` namespace.

When you have the XML Schema Instance namespace available, that is `"http://www.w3.org/2001/XMLSchema-instance"`, you can use the `schemaLocation` attribute. This attribute has two values:

The first value is the namespace to use.

The second value is the location of the XML schema to use for that namespace `"xsi:schemaLocation="message.xsd"`

3.1 :Validating xml against xsd

XML-Schema Definition

- Simple Element:
 - <xsd:element name="title" type="xsd:string"/>
- where "title" is the name of the element and "xsd:string" is the data type of the element
- Specifying default or fixed values:
 - <xsd:element name="title" type="xsd:string" default="No Title"/>
 - <xsd:element name="category" type="xsd:string" fixed="Common"/>



Copyright © Capgemini 2015. All Rights Reserved 14

Writing a Schema Definition for an XML File:

What is a Simple Element?

A simple element is an XML element that can contain only text. It cannot contain any other element or attribute. The text can be of many different types. It can be one of the types that are included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

You can also add restrictions to a data type in order to limit its content, and you can make it mandatory for the data to match a defined pattern.

Default and Fixed Values for Simple Elements:

Simple elements may have a default value or a fixed value specified.

A default value is automatically assigned to the element when no other value is specified. In the above example, default value is "No Title".

A fixed value is also automatically assigned to the element, and you cannot specify another value. In the above example, the fixed value is "common".

Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

Here are the corresponding simple element definitions:

```
<xsd:element name="lastname" type="xsd:string"/>
```

```
|  
<xs:element name="age" type="xs:integer"/>  
<xs:element name="dateborn" type="xs:date"/>
```

XML Schema Data Types

- XML Schema Data Types belongs to following categories:
 - XSD String: String data types are used for values that contains character strings.
 - XSD Date: Date and time data types are used for values that contain date and time.
 - XSD Numeric: Numeric data types are used for numeric values
 - XSD Misc: Other miscellaneous data types like boolean, base64Binary, hexBinary, float, double, etc.



XML Schema Vocabulary:

XML Schema has support for data types.

With the support for data types it is easier:

to describe permissible document content.

to validate the correctness of data.

to work with data from a database.

to define data patterns (data formats).

to convert data between different data types.

Apart from the above advantages, XML schemas are extensible. It implies that you can reuse your Schema in other Schemas and also reference multiple schemas from the same document.

String and Date Data Types

- String Data Type:
 - <xs:element name=" Author" type="xs:string"/>
 - <Author>John Smith</Author>
- NormalizedString Data Type:
 - <xs:element name="Author" type="xs:normalizedString"/>
 - <Author>John Smith</Author>
- Date Data Type:
 - <xs:element name="publishdate" type="xs:date"/>
 - < publishdate>2002-09-24</ publishdate>



String Data Types:

String Data Type:

The string data type can contain characters, line feeds, carriage returns, and tab characters.

NormalizedString Data Type

The normalizedString data type is derived from the String data type.

The normalizedString data type also contains characters, but the XML processor will remove line feeds, carriage returns, and tab characters.

Token Data Type

The token data type is also derived from the String data type.

The token data type also contains characters, but the XML processor will remove line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces.

Restrictions on String Data Types

Restrictions that can be used with String data types:

enumeration

length

maxLength

minLength

pattern

whiteSpace

Numeric and Boolean Data Types

- Decimal Data Type:

```
<xs:element name="price" type="xs:decimal"/>
<price>999.50</price> or
<price>+999.5450</price> or
<price>-999.5230</price>
```

- Integer Data Type:

```
<xs:element name="price" type="xs:integer"/>
<price>999</price> Or
<price>+999</price> Or
<price>-999</price>
```

- Boolean Data Type:

```
<xs:element name="disabled" type="xs:boolean"/>
<disabled>true</disabled>
```



Numeric Data Types:

Decimal Data Type: The decimal data type is used to specify a numeric value.

Integer Data Type: The integer data type is used to specify a numeric value without a fractional component.

Numeric Data Types: All the data types below derive from the Decimal data type (except for decimal itself)!

Byte: A signed 8-bit integer

Decimal: A decimal valueintA signed 32-bit integer

Integer: An integer valuelongA signed 64-bit integer

negativeInteger: An integer containing only negative values

(..,-2,-1)

nonNegativeInteger: An integer containing only non-negative values (0,1,2,..)

nonPositiveInteger: An integer containing only non-positive values (..,-2,-1,0)

positiveInteger: An integer containing only positive values (1,2,..)

Short: A signed 16-bit integer

unsignedLong: An unsigned 64-bit integer

unsignedInt: An unsigned 32-bit integer

unsignedShort: An unsigned 16-bit integer

unsignedByte: An unsigned 8-bit integer

3.1 :Validating xml against xsd

Attribute in XSD

- Defining an Attribute:

```
<xs:attribute name="AuthorID" type="xs:string"/>
```

where "AuthorID" is the name of the attribute and "xs:string" specifies the data type of the attribute.

- Creating Optional and Required Attributes:

```
<xs:attribute name="btype" type="xs:string" use="required"/>
```

Attributes are optional by default



Copyright © Capgemini 2015. All Rights Reserved 16

Attribute in XSD:

What is an Attribute?

Simple elements cannot have attributes. If an element has attributes, it is considered to be of complex type. However, the attribute itself is always declared as a simple type. This means that an element with attributes always has a complex type definition.

Example:

```
<Author AuthorID="A001">Smith</Author>
```

Following is a corresponding simple attribute definition:

```
<xs:attribute name="AuthorID" type="xs:string"/>
```

Default and Fixed Values for Attributes

Attributes may have a default value or a fixed value specified.

A default value is automatically assigned to the attribute when no other value is specified.

In the following example the default value is "UnKnown":

```
<xs:attribute name="AuthorID" type="xs:string" default="UnKnown"/>
```

Optional and Required Attributes

Attributes are optional, by default. To specify that the attribute is required, use the "use" attribute:

```
<xs:attribute name="AuthorID" type="xs:string" use="required"/>
```

Complex Type Element

- Illustration:

```
<xs:element name="book">
<xs:complexType>
<xs:sequence>
<xs:element name="title" type="xs:string"/>
<xs:element name="author" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```



Complex Element:

Complex Element:

A complex element is an XML element that contains other elements and/or attributes.

There are four kinds of complex elements:

Empty elements

Elements that contain only other elements

Elements that contain both – other elements and text

Elements that contain text

Examples of Complex XML Elements:

Example 1:

Consider a complex XML element, “product”, which is empty:

```
<product pid="1345"/>
```

It can be declared as:

```
<xs:element name="product">
<xs:complexType>
<xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
</xs:element>
```

Types of Indicators

- We have seven types of indicators:
 - Order indicators:
 - All
 - Choice
 - Sequence
 - Occurrence indicators:
 - maxOccurs
 - minOccurs
 - Group indicators:
 - Group name
 - attributeGroup name



Types of Indicators:

Indicators are specially used to control the occurrences of elements in different orders. Sometimes, we may want certain elements to occur only once, or certain elements may not be in a particular order, or certain elements may not be necessary at all (optional) and so on.

We can handle these kinds of issues by using indicators.

Order Indicators:

Order indicators are used to define how elements should occur.

All Indicator

- The <all> indicator specifies, by default, that the child elements can appear in any order and that each child element must occur once and only once

```
<xs:element name="book">
  <xs:complexType>
    <xs:all>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```



Indicator – Order, Occurrence, and Group:

In the example shown above, “book” element can have only “title & author” child elements which can occur in any sequence.

All Indicator:

When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1.

3.1 :Validating xml against xsd

Choice Indicator

- The <choice> indicator specifies that either one child element or another can occur

```
<xss:element name="person">
<xss:complexType>
<xss:choice>
<xss:element name="employee" type="employee"/>
<xss:element name="member" type="member"/>
</xss:choice>
</xss:complexType>
</xss:element>
```



Copyright © Capgemini 2015. All Rights Reserved 22

Sequence Indicator

- The <sequence> indicator specifies that the child elements must appear in a specific order

```
<xss:element name="book">
<xss:complexType>
<xss:sequence>
<xss:element name="title" type="xss:string"/>
<xss:element name="author" type="xss:string"/>
</xss:sequence>
</xss:complexType>
</xss:element>
```



maxOccurs Indicator

- The <maxOccurs> indicator specifies the maximum number of times an element can occur:

```
<x:element name="book">
<x:complexType>
<x:sequence>
<x:element name="title" type="x:string"/>
<x:element name="author" type="x:string"/>
<x:element name="vendor" type="x:string" maxOccurs="2"/>
</x:sequence>
</x:complexType>
</x:element>
```



Indicator – Order, Occurrence, and Group:

Occurrence Indicators:

Occurrence indicators are used to define how often an element can occur.

Note: For all “Order” and “Group” indicators (any, all, choice, sequence, group name, and group reference), the default value for maxOccurs and minOccurs is 1.

maxOccurs Indicator:

The <maxOccurs> indicator specifies the maximum number of times an element can occur.

minOccurs Indicator

- The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<x:element name="book">
<x:complexType>
  <x:sequence>
    <x:element name="title" type="x:string"/>
    <x:element name="author" type="x:string"/>
    <x:element name="vendor" type="x:string"      maxOccurs="2"
      minOccurs="0" />
  </x:sequence>
</x:complexType>
</x:element>
```



Indicator – Order, Occurrence, and Group:

minOccurs Indicator:

The example in the above slide indicates that the “vendor” element can occur a minimum of zero times and a maximum of two times in the “book” element.

Tip: To allow an element to appear for an unlimited number of times, use the maxOccurs="unbounded" statement.

3.1 :Validating xml against xsd

Group Indicators

- Group Indicators:
 - Group indicators are used to define related sets of elements
- Element Groups:
 - Element groups are defined with the group declaration, as shown below:

```
<xs:group name="groupname"> ... </xs:group>
```



Copyright © Capgemini 2015. All Rights Reserved 26

Indicator – Order, Occurrence, and Group:

Group Indicators:

You must define an all, choice, or sequence element inside the group declaration.

Group Indicators (Contd)

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence> </xs:group>
```

```
<xs:element name="person" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```



Group Indicators (Contd):

The example in the above slide defines a group named “persongroup”, that defines a group of elements that must occur in an exact sequence.

After you have defined a group, you can reference it in another group or complex type definition, as shown above.

Simple Type Element

- Illustration:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



Simple Element:

Simple Element:

A simple element is an XML element that contain text

Examples of Simple XML Elements:

Example 1:

Consider a simple XML element, “EmpName”, which contains the name of employee:

<EmpName>Smith</EmpName>

Restrictions on XSD Elements:

Restrictions on Content:

When an XML element or attribute has a datatype associated with , it puts a restriction on the element's or attribute's content. If an XML element is of type "xs:integer" and contains a string value "Nice Day", then the element will not validate.

With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called **facets**.

Some restrictions that apply on XSD elements are as follows:

| Constraint | Description |
|-------------------|--|
| Enumeration | Defines a list of acceptable values |
| FractionDigits | Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero. |
| Length | Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero. |
| MaxExclusive | Specifies the upper bounds for numeric values (the value must be less than this value) |
| MaxInclusive | Specifies the upper bounds for numeric values (the value must be less than or equal to this value) |
| MaxLength | Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero. |
| MinExclusive | Specifies the lower bounds for numeric values (the value must be greater than this value) |
| MinInclusive | Specifies the lower bounds for numeric values (the value must be greater than or equal to this value) |
| MinLength | Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero. |
| Pattern | Defines the exact sequence of characters that are acceptable |
| TotalDigits | Specifies the exact number of digits allowed. Must be greater than zero |
| WhiteSpace | Specifies the manner in which white space (line feeds, tabs, spaces, and carriage returns) is handled |

Restriction on Values

- Example

```
<xs:element name="Quantity">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="500"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



Restrictions on XSD Elements:

Restrictions on Values:

The example in the above slide defines an element called “Quantity” with a restriction. The value of book “Quantity” cannot be lower than 0 or greater than 500.

Restriction on Set Values

- Example

```
<xs:element name="Category">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="Dot Net"/>
<xs:enumeration value="BI"/>
<xs:enumeration value="RDBMS"/>
<xs:enumeration value="J2EE"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```



Restriction on Set of Values:

According to the example shown on the above slide, the Element category can have only four possible values which are Dot Net, BI, RDBMs, and J2EE.

3.2 : Simple Type restriction

Restrictions on Series of Values

- To limit the content of an XML element to define a series of numbers or letters that can be used, we can use the pattern constraint.

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- The only acceptable value is ONE of the LOWERCASE letters from a to z
- The “Category” element is a simple type with a restriction.
- The acceptable values are Dot Net, BI, RDBMS, and J2EE



Copyright © Capgemini 2015. All Rights Reserved 32

3.2 : Simple Type restriction

Restrictions on Series of Values

- Some more examples of Pattern

| | |
|--------------------------|--|
| [a-zA-Z][a-zA-Z][a-zA-Z] | THREE of the LOWERCASE OR UPPERCASE letters from a to z |
| [0-9]{10} | Any 10 digit number |
| [A-Z][0-9]{3} | 1 uppercase letter followed by 3 digits |
| [0-9][0-9][0-9][a-zA-Z]* | 3digits followed by any number of uppercase or lowercase letters |
| EMP[#_!] | 'EMP' followed by 1 # or ! Or _ |

- The “Category” element is a simple type with a restriction.
- The acceptable values are Dot Net, BI, RDBMS, and J2EE



Copyright © Capgemini 2015. All Rights Reserved 33

Restrictions on Series of Values

The above table shows how to give patterns in restrictions.

The next example defines an element called “gender” with a restriction. The only acceptable value is male or female:

```
<xs:element name="gender">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="male|female"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

The next example defines an element called “password” with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9:

```
<xs:element name="password">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="[a-zA-Z0-9]{8}"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Demo on XML-Schema Definition

- Demo on:
 - Shiporder.xsd (schema File)
 - Shiporder.xml (xml Document)
 - Validate the xml against xsd by Notepad++ with XML tools plugins



Copyright © Capgemini 2015. All Rights Reserved 34

Case Study

- Find the below xml file and create the xsd file to validate



Food.xml



Copyright © Capgemini 2015. All Rights Reserved 35

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<breakfast_menu>
    <food>
        <name>Belgian Waffles</name>
        <price>$5.95</price>
        <description>two of our famous Belgian
Waffles with plenty of real maple syrup</description>
        <calories>650</calories>
    </food>
    <food>
        <name>Strawberry Belgian Waffles</name>
        <price>$7.95</price>
        <description>light Belgian waffles covered
with strawberries and whipped cream</description>
        <calories>900</calories>
    </food>
    <food>
        <name>Berry-Berry Belgian Waffles</name>
        <price>$8.95</price>
        <description>light Belgian waffles covered
with an assortment of fresh berries and whipped cream</description>
        <calories>900</calories>
    </food>
</breakfast_menu>
```

Summary

- In this lesson, you have learnt:
 - A schema describes the arrangement of markup and character data within a valid XML document
 - XML Schema vocabulary defines different elements



Review Questions

- Question 1: List any four valid datatypes in XML Schema: ___, ___, ___, and ___.
- Question 2: The elements defined in a schema come from this namespace:
 - Option 1 : sourceNamespace
 - Option 2: targetNamespace
 - Option 3: cannot be specified
- Question 3: Choice is an Occurrence indicator.
 - True/False



Copyright © Capgemini 2015. All Rights Reserved 37

DBMS Concepts and SQL

Lesson 04:

Lesson Objectives

- To understand the following concepts
 - DDL commands (CREATE, ALTER and DROP)
 - Constraints
 - Sequence
 - DML command (Insert, Update, Delete)
 - Select Query, Joins and subquery



Table

- Tables are objects, which store the user data.
- Use the CREATE TABLE statement to create a table, which is the basic structure to hold data.
- For example:

```
CREATE TABLE book_master
(book_code number,
book_name varchar2(50),
book_pub_year number,
book_pub_author varchar2(50));
```



Creating tables is done with the create table command. You can add rows to a table with the INSERT statement, after creating a table.

The above create table command does the following:

Defines the table name

Defines the columns in the table and the datatypes of those columns

In above example, we create a table called BOOK_MASTER which has 4 columns. The first column and third column is defined as NUMBER datatype. This means we will be storing numbers in this column. The second and fourth columns are of VARCHAR2 datatype. We will be storing text data in these columns

Syntax:

```
CREATE TABLE table_name
(
{col_name.col_datatype [[CONSTRAINT
const_name][col_constraint]]},...
[table_constraint],...
)
[AS query]
```

If a table is created as shown in the slide, then there is no restriction on the data that can be stored in the table.

However, if we wish to put some restriction on the data, which can be stored in the table, then we must supply some “constraints” for the columns. We will

see the Constraints as next topic.

ALTER Table

- Given below is an example of ALTER TABLE:

```
ALTER TABLE table_name
[ADD (col_name col_datatype col_constraint ,...)]]
[ADD (table_constraint)]
[DROP CONSTRAINT constraint_name]
[MODIFY existing_col_name new_col_datatype
    new_constraint new_default]
[DROP COLUMN existing_col_name]
[SET UNUSED COLUMN existing_col)name];
```



Examples of ALTER TABLE:

Table_name must be an existing table.

A column can be removed from an existing table by using ALTER TABLE.

The uses of modifying columns are:

Can increase the width of a character column, any time.

Can increase the number of digits in a number, any time.

Can increase or decrease the number of decimal places in a number column, any time. Any reduction on precision and scale can be on empty columns only.

Can add only NOT NULL constraint by using Column constraints. All other constraints have to be specified as Table constraints.

4.1 DDL commands (CREATE, ALTER and DROP)

ALTER Table – Add clause

- The “Add” keyword is used to add a column or constraint to an existing table.
- For adding three more columns to the emp table, refer the following example:

```
ALTER TABLE Student_Master  
ADD (last_name varchar2(25));
```



Copyright © Capgemini 2015. All Rights Reserved 5

ALTER TABLE – Add clause:

The ADD clause allows to add a column or constraint. You can also add multiple columns in one statement separated by comma.

A column with constraints can also be added as shown in the following example:

```
ALTER TABLE Department_Master  
ADD (dept_name varchar2(10) NOT NULL);
```

4.1 DDL commands (CREATE, ALTER and DROP)

ALTER Table – Add clause

- For adding Referential Integrity on "mgr_code" column, refer the following example:

```
ALTER TABLE staff_master  
ADD CONSTRAINT FK FOREIGN KEY (mgr_code) REFERENCES  
staff_master(staff_code);
```



Copyright © Capgemini 2015. All Rights Reserved 6

ALTER Table – MODIFY clause

- **MODIFY clause:**

- The “Modify” keyword allows making modification to the existing columns of a table.
 - For Modifying the width of “sal” column, refer the following example:

```
ALTER TABLE staff_master  
MODIFY (staff_sal number (12,2) );
```



ALTER TABLE- Modify Clause

The use of modifying the columns with the Enable | Disable clause are:

Can increase “column width” of a character any time.

Can increase the “number of digits” in a number at any time.

Can increase or decrease the “number of decimal places” in a number column at any time. Any reduction on “precision” and “scale” can only be on empty columns.

Can only add the NOT NULL constraint by using “column constraints”. Rest all other constraints have to be specified as “table constraints”.

4.1 DDL commands (CREATE, ALTER and DROP)

ALTER Table – Enable | Disable clause

- **ENABLE | DISABLE Clause:**

- The ENABLE | DISABLE clause allows constraints to be enabled or disabled according to the user choice without removing them from a table.
- Refer the following example:

```
ALTER TABLE staff_master DISABLE CONSTRAINT  
SYS_C000934;
```



Copyright © Capgemini 2015. All Rights Reserved 6

4.1 DDL commands (CREATE, ALTER and DROP)

ALTER Table – DROP clause

- The DROP clause is used to remove constraints from a table.
 - For Dropping the FOREIGN KEY constraint on “department”, refer the following example:

```
ALTER TABLE student_master  
DROP CONSTRAINT stu_dept_fk ;
```



Copyright © Capgemini 2015. All Rights Reserved 9

ALTER TABLE – Drop Clause

To remove the PRIMARY KEY constraint on the DEPARTMENT table and drop the associated FOREIGN KEY constraint on the DEPT_CODE column.

```
ALTER TABLE department_master  
DROP PRIMARY KEY CASCADE;
```

Dropping Column

- Given below are the ways for “Dropping” a column:
 - 1a. Marking the columns as unused and then later dropping them.
 - 1b. The following command can be used later to permanently drop the columns.

```
ALTER TABLE staff_master SET UNUSED COLUMN staff_address;
ALTER TABLE staff_master SET UNUSED (staff_sal, hiredate);
```

```
ALTER TABLE emp DROP UNUSED COLUMNS;
```



Ways for “Dropping” a column:

Marking the columns as “Unused” and then later dropping them:

Oracle onwards a new feature to “drop” and “set” the “unused columns” in a table is added

First command as shown in the slide, allows you to mark a column as unused and

Second command as shown in the following slide, lets you drop the unused column from the table to create more free space.

This feature drops the column from a table and releases any space back to the segment.

Note that:

Columns once marked as unused cannot be recovered.

Marking the columns as unused does not release the space occupied by them back to the database.

Until you actually drop these columns, they continue to count towards the absolute limit of 1000 columns per table.

If you mark a column of data type LONG as UNUSED, you cannot add another LONG column to the table until you actually drop the unused LONG column.

The advantage of the marking column as “unused” and then dropping them is that marking the columns is much faster process than dropping the columns.

You can refer to the data dictionary table USER_UNUSED_COL_TABS to get information regarding the tables with columns marked as unused.

Dropping Column

- Directly dropping the columns.

```
ALTER TABLE staff_master DROP COLUMN staff_sal;
```



Ways for “Dropping” a column (contd.):

DROP COLUMN:

This feature allows directly dropping the column.

For DROP COLUMN command shown in the slide, to work successfully, the table should be exclusively locked by the user by giving the command.

All “indexes” defined on any of the target columns are also dropped.

All “constraints” that reference a target column are removed.

Note that this command should be used with caution.

The CASCADE CONSTRAINTS clause is used along with the DROP COLUMN clause.

The CASCADE CONSTRAINTS clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.

The CASCADE CONSTRAINTS clause also drops all multicolumn constraints defined on the dropped columns

Drop a Table

- The DROP TABLE command is used to remove the definition of a table from the database.
- For Example:

```
DROP TABLE staff_master;
```

```
DROP TABLE Department_master  
CASCADE CONSTRAINTS;
```



Deleting Database Objects: Tables

Deleting objects that exist in the database is an easy task. Just say:

```
DROP Obj_Type obj_name;
```

A table that is dropped cannot be recovered. When a table is dropped, dependent objects such as indexes are automatically dropped. Synonyms and views created on the table remain, but give an error if they are referenced.

You cannot delete a table that is being referenced by another table. To do so use the following:

```
DROP TABLE table-name CASCADE CONSTRAINTS;
```

What is Data Integrity?

- Data Integrity:

- “Data Integrity” allows to define certain “data quality requirements” that must be met by the data in the database.
- Oracle uses “Integrity Constraints” to prevent invalid data entry into the base tables of the database.
 - You can define “Integrity Constraints” to enforce the business rules you want to associate with the information in a database.
 - If any of the results of a “DML statement” execution violate an “integrity constraint”, Oracle rolls back the statement and returns an error.



Copyright © Capgemini 2015. All Rights Reserved 13

Data Integrity: Integrity Constraint

An Integrity Constraint is a declarative method of defining a rule for a column of a table.

Example of Data Integrity:

Assume that you define an “Integrity Constraint” for the Staff_Sal column of the Staff_Master table.

This Integrity Constraint enforces the rule that “no row in this table can contain a numeric value greater than 10,000 in this column”.

If an INSERT or UPDATE statement attempts to violate this “Integrity Constraint”, Oracle rolls back the statement and returns an “information error” message.

Advantages

- Advantages of Integrity Constraints:

- Integrity Constraints have advantages over other alternatives. They are:
 - Enforcing “business rules” in the code of a database application.
 - Using “stored procedures” to completely control access to data.
 - Enforcing “business rules” with triggered stored database procedures.



Applying Constraints

- Constraints can be defined at
 - Column Level

```
CREATE TABLE tablename  
(column datatype [DEFAULT expr] [column_constraint] ,  
.....)
```

- Table Level

```
CREATE TABLE tablename  
(column datatype,  
column datatype  
.....  
[CONSTRAINT constraint_name] constraint_type (column,...))
```



Copyright © Capgemini 2015. All Rights Reserved 15

Applying Constraints:

In Oracle you can apply constraints

Column Level - References a single column and is defined within a specification for the owning column; can be any type of integrity constraint

Table Level - References one or more columns and is defined separately from the definitions of the columns in the table; can define any constraints except NOT NULL

Types of Integrity Constraints

- Let us see the types of Data Integrity Constraints:
 - Nulls
 - Default
 - Unique Column Values
 - Primary Key Values
 - Check
 - Referential Integrity



Types of Data Integrity:

Oracle supports the following Integrity Constraints:

NOT NULL constraints for the rules associated with nulls in a column. “Null” is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a “null” value (the absence of a value) in that column.

UNIQUE key constraints for the rule associated with unique column values. A “unique value” rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a “unique value” in that column (or set of columns).

PRIMARY KEY constraints for the rule associated with primary identification values. A “primary key” value rule defined on a key (a column or set of columns) specifies that “each row in the table can be uniquely identified by the values in the key”.

FOREIGN KEY constraints for the rules associated with referential integrity. A “Referential Integrity” rule defined on a key (a column or set of columns) in one table guarantees that “the values in that key, match the values in a key in a related table (the referenced value)”.

Oracle currently supports the use of FOREIGN KEY integrity constraints to define the referential integrity actions, including:

update and delete No Action

delete CASCADE

delete SET NULL

CHECK constraints for complex integrity rules

NOT NULL Constraint

- The user will not be allowed to enter null value.
- For Example:
 - A NULL value is different from a blank or a zero. It is used for a quantity that is “unknown”.
 - A NULL value can be inserted into a column of any data type.

```
CREATE TABLE student_master
(student_code number(4) NOT NULL,
dept_code number(4) CONSTRAINT dept_code_nn
NOT NULL);
```



NOT NULL constraint:

Often there may be records in a table that do not have values for every field.

This could be because the information is not available at the time of the data entry or because the field is not applicable in every case.

If the column is created as **NULLABLE**, in the absence of a user-defined value the DBMS will place a **NULL** value in the column.

A **NULL** value is different from a blank or a zero. It is used for a quantity that is “unknown”.

“Null” is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a “null” value (the absence of a value) in that column.

A **NULL** value can be inserted into a column of any data type.

Principles of NULL values:

Setting a **NULL** value is appropriate when the “actual value” is unknown, or when a value is not meaningful.

A **NULL** value is not equivalent to the value of “zero” if the data type is **number**, and it is not equivalent to “spaces” if the data type is **character**.

A **NULL** value will evaluate to **NULL** in any expression

For example: **NULL** multiplied by 10 is **NULL**.

NULL value can be inserted into columns of any data type.

If the column has a **NULL** value, Oracle ignores any **UNIQUE**, **FOREIGN KEY**, and **CHECK** constraints that may be attached to the column.

DEFAULT clause

- If no value is given, then instead of using a “Not Null” constraint, it is sometimes useful to specify a default value for an attribute.
- For Example:
 - When a record is inserted the default value can be considered.

```
CREATE TABLE staff_master(  
Staff_Code number(8) PRIMARY KEY,  
Staff_Name varchar2(50) NOT NULL,  
Staff_dob date,  
Hiredate date DEFAULT sysdate,  
.....)
```



UNIQUE constraint

- The keyword UNIQUE specifies that no two records can have the same attribute value for this column.
- For Example:

```
CREATE TABLE student_master  
(student_code number(4),  
 student_name varchar2(30) ,  
CONSTRAINT stu_id_uk UNIQUE(student_code )) ;
```



UNIQUE constraint:

The UNIQUE constraint does not allow duplicate values in a column.

If the UNIQUE constraint encompasses two or more columns, then two equal combinations are not allowed.

However, if a column is not explicitly defined as NOT NULL, then NULLS can be inserted multiple number of times.

A UNIQUE constraint can be extended over multiple columns.

A “unique value” rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a “unique value” in that column (or set of columns).

PRIMARY KEY constraint

- The Primary Key constraint enables a unique identification of each record in a table.
- For Example:

```
CREATE TABLE Staff Master
(staff_code number(6)
CONSTRAINT staff_id_pk PRIMARY KEY,
staff_name varchar2(20)
.....);
```



PRIMARY KEY constraint:

On a technical level, a PRIMARY KEY combines a UNIQUE constraint and a NOT NULL constraint.

Additionally, a table can have at the most one PRIMARY KEY.

After creating a PRIMARY KEY, it can be referenced by a FOREIGN KEY.

A “primary key” value rule defined on a key (a column or set of columns) specifies that “each row in the table can be uniquely identified by the values in the key”.

The example on the slide defines the primary key constraint at the column level. The same example is seen below with the constraint defined at table level

```
CREATE TABLE Staff Master
(staff_code number(6) ,
staff_name varchar2(20),
.....)
CONSTRAINT staff_id_pk PRIMARY KEY
(staff_code))
;
```

CHECK constraint

- CHECK constraint allows users to restrict possible attribute values for a column to admissible ones.
- For Example:

```
CREATE TABLE staff_master  
( staff_code number(2),  
  staff_name varchar2(20),  
  staff_sal  number(10,2) CONSTRAINT staff_sal_min  
              CHECK (staff_sal >1000),  
.....);
```



CHECK constraint:

A CHECK constraint allows to state a minimum requirement for the value in a column.

If more complicated requirements are desired, an INSERT trigger must be used.

FOREIGN KEY constraint

- The FOREIGN KEY constraint specifies a “column” or a “list of columns” as a foreign key of the referencing table.
- The referencing table is called the “child-table”, and the referenced table is called “parent-table”.
- For Example:

```
CREATE TABLE student_master
(student_code number(6) ,
dept_code number(4) CONSTRAINT stu_dept_fk
    REFERENCES department_master(dept_code),
student_name varchar2(30));
```



FOREIGN KEY Constraint Keywords:

Given below are a few Foreign Key constraint keywords:

FOREIGN KEY: Defines the column in the child table at the table constraint level.

REFERENCES: Identifies the table and column in the parent table.

ON DELETE CASCADE: Deletes the dependent rows in the child table when a row in the parent table is deleted.

ON DELETE SET NULL: Converts dependent FOREIGN KEY values to NULL.

You can query the **USER_CONSTRAINTS** table to view all constraint definitions and names.

You can view the columns associated with the constraint names in the **USER_CONS_COLUMNS** view.

In EMP table, for deptno column, if we want to allow only those values that already exist in deptno column of the DEPT table, we must enforce what is known as REFERENTIAL INTEGRITY. To enforce REFERENTIAL INTEGRITY, declare deptno field of DEPT table as PRIMARY KEY, and deptno field of EMP table as FOREIGN KEY as follows

Usage of Sequence

- A “Sequence” is an object, which can be used to generate sequential numbers.
- A Sequence is used to fill up columns, which are declared as UNIQUE or PRIMARY KEY.
- A Sequence uses “NEXTVAL” to retrieve the next value in the sequence order.



Sequence:

A Sequence:

automatically generates unique numbers.

is a “shareable object”.

is typically used to create a PRIMARY KEY value.

replaces application code.

speeds up the efficiency of accessing sequence values when cached in memory.

Creating a Sequence

- For example, suppose we have created a sequence "seq_no", then its next value can be obtained as "seq_no.nextval".

```
CREATE SEQUENCE seq_name
[INCREMENT BY n1] [START WITH n2]
[MAXVALUE n3] [MINVALUE n4] [CYCLE|NOCYCLE]
[CACHE|NOCACHE];
```



Sequence:

In the example shown in the slide:

START WITH indicates the first NUMBER in the series.

INCREMENT BY is the difference between consecutive numbers. If n1 is negative, then the numbers that are generated are in a descending order.

MAXVALUE and MINVALUE indicate the extreme values.

CYCLE indicates that once the extreme is reached, it starts the cycle again with n2.

NOCYCLE means that once the extreme is reached, it stops generating numbers.

CACHE caches the specified number of sequence values in the memory. This speeds access, but all cached numbers are lost when the database is shut down. The default value is 20

Confirming Sequences

As shown below, you need to verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SELECT sequence_name, min_value, max_value,
       increment_by, last_number
  FROM user_sequences;
```

The LAST_NUMBER column displays the next available sequence number if

DBMS Concepts and SQL

NOCACHE is specified.

Creating a Sequence

- Here is one more example of sequence:
 - s1 will generate numbers 1,2,3....,10000, and then stop.

```
CREATE SEQUENCE s1  
INCREMENT BY 1  
START WITH 1  
MAXVALUE 10000  
NOCYCLE ;
```



NEXTVAL and CURRVAL pseudo columns

- NEXTVAL returns the next available sequence value.
 - It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for the Sequence before CURRVAL can be referenced.



Referencing a Sequence:

After you create a Sequence, it generates sequential numbers that can be used in your tables. You can reference the Sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

NEXTVAL and CURRVAL Pseudocolumns:

The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified Sequence. You must qualify NEXTVAL with the sequence name. When you reference sequence.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer a Sequence number that the current user has just generated.

NEXTVAL must be used to generate a sequence number in the session of the current user, before CURRVAL can be referenced.

You must qualify CURRVAL with the sequence name.

When “sequence.CURRVAL” is referenced, the last value returned to that user’s process is displayed.

Drop a Sequence

- A Sequence can be removed from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the Sequence can no longer be referenced.

```
DROP SEQUENCE dept_deptid_seq;  
Sequence dropped.
```



Removing a Sequence

To remove a sequence from the data dictionary, use the DROP SEQUENCE statement.

To remove the Sequence, you must be the owner of the Sequence or possess the DROP ANY SEQUENCE privilege.

Syntax:

```
DROP SEQUENCE sequence;
```

where: *sequence* is the name of the sequence generator.

For more information, refer Oracle9i SQL Reference, “DROP SEQUENCE”.

contd.

Data Manipulation Language

- Data Manipulation Language (DML) is used to perform the following routines on database information:
 - Retrieve
 - Insert
 - Modify
- DML changes data in an object. If you insert a row into a table, that is DML.
- All DML statements change data, and must be committed before the change becomes permanent.



INSERT

- **INSERT command:**

- INSERT is a DML command. It is used to add rows to a table.
- In the simplest form of the command, the values for different columns in the row to be inserted have to be specified.
- Alternatively, the rows can be generated from some other tables by using a SQL query language command.



Addition of Data into Tables:

Requisites for using INSERT command:

If values are specified for all columns in the order specified at creation, then col_names could be omitted.

Values should match “data type” of the respective columns.

Number of values should match the number of column names mentioned.

All columns declared as NOT NULL should be supplied with a value.

Character strings should be enclosed in quotes.

Date values should be enclosed in quotes.

Values will insert one row at a time.

Query will insert all the rows returned by the query.

The table_name can be a “table” or a “view”. If table_name is a “view”, then the following restrictions apply:

The “view” cannot have a GROUP BY, CONNECT BY, START WITH, DISTINCT, UNION, INTERSECT, or MINUS clause or a join.

If the “view” has WITH CHECK OPTION clause, then a row, which will not be returned by the view, cannot be inserted.

4.4 DML command (Insert, Update, Delete)

Inserting Rows into a Table

- Inserting by specifying values:
- Example: To insert a new record in the DEPT table

```
INSERT INTO table_name[(col_name1,col_name2,...)]
{VALUES (value1,value2,...) | query};
```

```
INSERT INTO Department_master
VALUES (10, 'Computer Science');
```



Copyright © Capgemini 2015. All Rights Reserved 32

Inserting Rows into a Table:

Example:

Inserting a row in EMP table giving all values.

```
INSERT INTO student_master
VALUES(1001,'Amit',10,'11-Jan-80','Chennai');
```

10 is a dept number which exists in DEPARTMENT_MASTER table

Inserting a row in STAFF_MASTER table giving some values.

```
INSERT INTO staff_master
(staff_code,staff_name,design_code,dept_code)
VALUES(100001,'Arvind',102,30);
```

This row will be created if all the constraints like NOT NULL are satisfied.

4.4 DML command (Insert, Update, Delete)

Inserting Rows into a Table

- Inserting by using “substitution variables”:
- Example: In the example given below, when the command is run, values are prompted every time.

```
INSERT INTO department_master  
VALUES (&dept_code, '&dept_name');  
Enter a value for dept_code : 20  
Enter a value for dept_name : Electricals
```



Copyright © Capgemini 2015. All Rights Reserved 33

Inserting Rows into a Table:

Inserting by using “substitution variables”:

The problem with the INSERT statement is that it adds only “one row” to the table.

However, by using “substitution variables” the speed of data input can be increased.

Whenever a “substitution variable” is placed in a “value” field, the user will be prompted to enter the “actual value” when the command is executed.

4.4 DML command (Insert, Update, Delete)

DELETE

- The DELETE command is used to delete one or more rows from a table.
 - The DELETE command removes all rows identified by the WHERE clause.

```
DELETE [FROM] {table_name | alias }  
[WHERE condition];
```



Copyright © Capgemini 2015. All Rights Reserved 34

Deletion of Data from Tables

The table_name can be a “table” or a “view”.

The DELETE command is used to delete one or more rows from a table.

The DELETE statement removes all rows identified by the WHERE clause.

This is another DML, which means we can rollback the deleted data, and that to make our changes permanent.

If WHERE clause is omitted, all rows from the table are removed. Else all rows which satisfy the condition are removed.

FROM clause can be omitted without affecting the statement.

4.4 DML command (Insert, Update, Delete)

Deleting Rows from Table

- Example 1: If the WHERE clause is omitted, all rows will be deleted from the table.
- Example 2: If we want to delete all information about department 10 from the Emp

```
DELETE  
FROM staff_master;
```

```
DELETE  
FROM student_master  
WHERE dept_code=10;
```



Copyright © Capgemini 2015. All Rights Reserved 35

Deletion of Data from Tables

Example 3:

```
DELETE staff_master WHERE staff_name = 'Anil';
```

UPDATE

- Use the UPDATE command to change single rows, groups of rows, or all rows in a table.
 - In all data modification statements, you can change the data in only “one table at a time”.

```
UPDATE table_name  
SET col_name = value|  
    col_name = SELECT_statement_returning_single_value|  
    (col_name,...) = SELECT_statement  
[WHERE condition];
```



Modifying / Updating existing Data in a Table:

The table_name can be a “table” or a “view”.

The “value” can be a value, an expression, or a query, which returns a single value.

The UPDATE command provides automatic navigation to the data.

Note: If the WHERE clause is omitted, all rows in the table will be updated by a value that is currently specified for the field. Else only those rows which satisfy the condition will be updated.

4.4 DML command (Insert, Update, Delete)

Updating Rows from Table

- Example 1: To UPDATE the column “dname” of a row, where deptno is 10, give the following command:

```
UPDATE department_master  
SET dept_name= 'Information Technology'  
WHERE dept_code=10;
```



Copyright © Capgemini 2015. All Rights Reserved 37

4.4 DML command (Insert, Update, Delete)

Updating Rows from Table

- Example 2: To UPDATE the subject marks details of a particular student, give the following command:

```
UPDATE student_marks  
SET subject1= 80 , subject2= 70  
WHERE student_code=1005;
```



Copyright © Capgemini 2015. All Rights Reserved 38

The Select Statement and Syntax

- The SELECT command is used to retrieve rows from a single table or multiple Tables or Views.
- A query may retrieve information from specified columns or from all of the columns in the Table.
- It helps to select the required data from the table.

```
SELECT [ALL | DISTINCT] { * | col_name,...}
FROM table_name alias, ...
[ WHERE expr1 ]
[ CONNECT BY expr2 [ START WITH expr3 ] ]
[ GROUP BY expr4 ] [ HAVING expr5 ]
[ UNION | INTERSECT | MINUS SELECT ... ]
[ ORDER BY expr | ASC | DESC ];
```



The SELECT Statement:

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set). The statement begins with the SELECT keyword. The basic SELECT statement has three clauses:

```
SELECT
FROM
WHERE
```

The SELECT clause specifies the table columns that are retrieved.

The FROM clause specifies the tables accessed.

The WHERE clause specifies which table rows are used. The WHERE clause is optional; if missing, all table rows are used.

Note:

Each clause is evaluated on the result set of a previous clause. The final result of the query will be always a “result table”.

Only FROM clause is essential. The clauses WHERE, GROUP BY, HAVING, ORDER BY are optional.

All the examples that follow are based on EMP and DEPT tables that are already available.

4.5 Select Query, Joins and subquery

Selecting Columns

- Displays all the columns from the student_master table

```
SELECT *  
FROM student_master;
```

- Displays selected columns from the student_master table

```
SELECT student_code, student_name  
FROM student_master;
```



Copyright © Capgemini 2015. All Rights Reserved 40

The WHERE clause

- The WHERE clause is used to specify the criteria for selection.
 - For example: displays the selected columns from the student_master table based on the condition being satisfied

```
SELECT student_code, student_name, student_dob
      FROM student_master
     WHERE dept_code = 10;
```



The WHERE Clause:

The WHERE clause is used to perform “selective retrieval” of rows. It follows the FROM clause, and specifies the search condition.

The result of the WHERE clause is the row or rows retrieved from the Tables, which meet the search condition.

The clause is of the form:

```
WHERE <search condition>
```

Comparison Predicates:

The Comparison Predicates specify the comparison of two values.

It is of the form:

$$<\text{Expression}> <\text{operator}> <\text{Expression}>$$

$$<\text{Expression}> <\text{operator}> <\text{subquery}>$$

The operators used are shown on the next slide:

contd.

The AS clause

- The AS clause is used to specify an alternate column heading.
- For example: displays the selected columns from the student_master table based on the condition being satisfied. Observe the column heading

```
SELECT student_dob as "Date of Birth"  
      FROM student_master  
     WHERE dept_code = 10;  
  
-- quotes are required when the column heading contains a space
```

```
SELECT student_dob "Date of Birth"  
      FROM student_master  
     WHERE dept_code = 10;  
  
-- AS keyword is optional
```



The AS Clause:

The AS clause is used to give a different column heading (other than column name) to one or more columns used in the select statement. It follows the column name, and can be used for one or more columns.

The AS keyword is optional.

The clause is of the form:

```
Select column1 heading1, column2 as heading1,  
column3 as "heading3 contains space" from  
table_name
```

Character Strings and Dates

- Are enclosed in single quotation marks
- Character values are case sensitive
- Date values are format sensitive

```
SELECT student_code, student_dob  
      FROM student_master  
     WHERE student_name = 'Sunil' ;
```



Oracle Database store dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds.

The date datatype is covered in detail later.

Mathematical, Comparison & Logical Operators

- Mathematical Operators:

- Examples: +, -, *, /

- Comparison Operators:

| Operator | Meaning |
|---------------|--------------------------|
| = | Equal to |
| > | Greater than |
| >= | Greater than or Equal to |
| < | Less than |
| <= | Less than or Equal to |
| <>, !=, or ^= | Not Equal to |

- Logical Operators:

- Examples: AND, OR, NOT



Operators:

Operators are used in “expressions” or “conditional statements”. They show equality, inequality, or a combination of both.

Operators are of three types:

mathematical

logical

range (comparison)

These operators are mainly used in the WHERE clause, HAVING clause in order to filter the data to be selected.

Mathematical operators:

These operators add, subtract, multiply, divide, and compare equality of numbers and strings. They are +, -, *, /

Comparison Operators:

These operators are used to compare the column data with specific values in a condition. “Comparison Operators” are also used along with the “SELECT statement” to filter data based on specific conditions. The table in the slide describes each Comparison operator. Comparison operators indicate how the data should relate to the given search value.

Logical Operators:

There are three Logical Operators namely AND, OR and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output or not. When retrieving data by using a SELECT statement, you can use logical operators in the WHERE clause. This allows you to combine more than one condition.

Other Comparison Operators

| Other Comparison operators | Description |
|----------------------------|---|
| [NOT] BETWEEN x AND y | Allows user to express a range. For example: Searching for numbers BETWEEN 5 and 10. The optional NOT would be used when searching for numbers that are NOT BETWEEN 5 AND 10. |
| [NOT] IN(x,y,...) | Is similar to the OR logical operator. Can search for records which meet at least one condition contained within the parentheses. For example: Pubid IN (1, 4, 5), only books with a publisher id of 1, 4, or 5 will be returned. The optional NOT keyword instructs Oracle to return books not published by Publisher 1, 4, or 5. |



Other Comparison Operators

| Other Comparison operators | Description |
|----------------------------|--|
| [NOT] LIKE | Can be used when searching for patterns if you are not certain how something is spelt. For example: title LIKE 'TH%'. Using the optional NOT indicates that records that do contain the specified pattern should not be included in the results. |
| IS[NOT]NULL | Allows user to search for records which do not have an entry in the specified field. For example: Shipdate IS NULL. If you include the optional NOT, it would find the records that do not have an entry in the field. For example: Shipdate IS NOT NULL. |



BETWEEN ... AND Operator

- The BETWEEN ... AND operator finds values in a specified range:

```
SELECT staff_code,staff_name  
      FROM staff_master  
     WHERE staff_dob BETWEEN '01-Jan-1980'  
           AND '31-Jan-1980';
```



IN Operator

- The IN operator matches a value in a specified list.
 - The List must be in parentheses.
 - The Values must be separated by commas.

```
SELECT dept_code  
      FROM department_master  
     WHERE dept_name IN ('Computer Science', 'Mechanics');
```



IN predicate:

It is of the form:

<Expression> IN <LIST>
<Expression> IN <SUBQUERY>

The data types should match.

LIKE Operator

- The LIKE operator performs pattern searches.
 - The LIKE operator is used with wildcard characters.
 - Underscore (_) for exactly one character in the indicated position
 - Percent sign (%) to represent any number of characters

```
SELECT book_code,book_name
  FROM book_master
 WHERE book_pub_author LIKE '%Kanetkar%' ;
```



LIKE predicate:

It is of the form:

<COLUMN> LIKE < PATTERN >

The pattern contains a search string along with other special characters % and _. The % character represents a string of any length where as _ (underscore) represents exactly one character.

A pattern %XYZ% means search has to be made for string XYZ in any position. A pattern '_XYZ%' means search has to be made for string XYZ in position 2 to 4.

To search for characters % and _ in the string itself we have to use an "escape" character.

For example: To search for string NOT_APP in column status, we have to use the form Status like 'NOT_APP' ESCAPE '\'

The use of \ as escape character is purely arbitrary.

|| Operator (Concatenation)

- The || operator performs concatenation.
 - between a string literal and a column name.
 - between two column names
 - between string literal and a pseudocolumn

```
SELECT 'Hello' || student_name
FROM student_master
```

-- only single quotes not double

```
SELECT student_code || ' ' || student_name
FROM student_master
```

```
SELECT 'Today is ' || sysdate
FROM dual
```



Retrieval of Constant values by using Dual Table

A “dual” is a table, which is created by Oracle along with the data dictionary. It consists of exactly one column, whose name is dummy, and one record. The value of that record is X.

```
SQL>desc dual;
Name          Null?    Type
DUMMY        VARCHAR2(1)
SQL>Select * from dual;
D
-
X
```

The owner of dual is SYS. However, “dual” can be accessed by every user.

As “dual” contains exactly one row (unless someone has fiddled with it), it is guaranteed to return exactly one row in SELECT statements.

```
SQL>select sysdate from dual;
```

For example, you can use it for math:

```
SQL>SELECT (319/212)+10 FROM DUAL;
```

And, you can use it to increment sequences:

```
SQL>SELECT employee_seq.NEXTVAL FROM
DUAL;
```

Logical Operators

- Logical operators are used to combine conditions.
 - Logical operators are NOT, AND, OR.
 - NOT reverses meaning.
 - AND both conditions must be true.
 - OR at least one condition must be true.
 - Use of AND operator

```
SELECT staff_code,staff_name,staff_sal  
      FROM staff_master  
     WHERE dept_code = 10  
       AND staff_dob > '01-Jan-1945';
```



The AND operator displays a record if both the first condition and the second condition is true.

One More Example:

```
SQL> SELECT title, pubid, category  
2   FROM books  
3  WHERE pubid = 3  
4  AND category = 'COMPUTER';
```

Combining Predicates by using Logical Operators:

The predicates can be combined by using logical operators like AND, OR, NOT. The evaluation proceeds from left to right and order of evaluation is:

* Enclosed in parenthesis

AND

OR

Using AND or OR Clause

- Use of OR operator:

```
SELECT book_code  
      FROM book_master  
     WHERE book_pub_author LIKE '%Kanetkar%'  
       OR book_name LIKE '%Pointers%';
```



The OR operator displays a record if either the first condition or the second condition is true.

You can also combine AND and OR as shown in above example. (use parenthesis to form complex expressions).

Using NOT Clause

- The NOT operator finds rows that do not satisfy a condition.
 - For example: List staff members working in depts other than 10 & 20.

```
SELECT staff_code,staff_name  
      FROM staff_master  
     WHERE dept_code NOT IN ( 10,20 );
```



Note: NOT is a negation operator.

Treatment of NULL Values

- NULL is the absence of data.
- Treatment of this scenario requires use of IS NULL operator.

```
SQL>SELECT student_code  
      FROM student_master  
     WHERE dept_code IS NULL;
```



NULL predicate:

The NULL predicate specifies a test for NULL values. The form for NULL predicate is:

< COLUMN SPECIFICATION > IS NULL.

< COLUMN SPECIFICATION > IS NOT NULL.

< COLUMN SPECIFICATION > IS NULL returns TRUE only when column has NULL values.

<COLUMN> = NULL cannot be used to compare null values.

Operator Precedence

- Operator precedence is decided in the following order:

| Levels | Operators |
|--------|---|
| 1 | * (Multiply), / (Division), % (Modulo) |
| 2 | + (Positive), - (Negative), + (Add), (+ Concatenate), - (Subtract), & (Bitwise AND) |
| 3 | =, >, <, >=, <=, <>, !=, !=, !< (Comparison operators) |
| 4 | NOT |
| 5 | OR |
| 6 | AND |
| 7 | ALL, ANY, BETWEEN, IN, LIKE, OR, SOME |
| 8 | = (Assignment) |



Copyright © Capgemini 2015. All Rights Reserved 55

Operator Precedence:

When a complex expression has multiple operators, the operator precedence (or order of execution of operators) determines the sequence in which the operations are performed.

The order of execution can significantly affect the resulting value.

The operators have the precedence levels as shown in the table given in the slide.

An operator on higher levels is evaluated before an operator on lower level.

The ORDER BY clause

- The ORDER BY clause presents data in a sorted order.
 - It uses an “ascending order” by default.
 - You can use the DESC keyword to change the default sort order.
 - It can process a maximum of 255 columns.
- In an ascending order, the values will be listed in the following sequence:
 - Numeric values
 - Character values
 - NULL values
- In a descending order, the sequence is reversed.



The Order By Clause:

- A query with its various clauses (FROM, WHERE, GROUP BY, HAVING) determines the rows to be selected and the columns. The order of rows is not fixed unless an ORDER BY clause is given.
- An ORDER BY clause is of the form:

ORDER BY < Sort list> ASC/DESC

- The columns to be used for ordering are specified by using the “column names” or by specifying the “serial number” of the column in the SELECT list.
- The sort is done on the column in “ascending” or “descending” order. By default the ordering of data is “ascending” order.

contd.

Sorting Data

- The output of the SELECT statement can be sorted using ORDER BY clause
 - ASC : Ascending order, default
 - DESC : Descending order
- Display student details from student_master table sorted on student_code in descending order.

```
SELECT Student_Code,Student_Name,Dept_Code, Student_dob  
      FROM Student_Master  
      ORDER BY Student_Code DESC ;
```



SQL: 1999 Compliant Joins - Syntax

- Syntax

```
SELECT table1.column, table2.column  
FROM table1  
[CROSS JOIN table2] |  
[NATURAL JOIN table2] |  
[JOIN table2 USING (column_name)] |  
[JOIN table2 ON (table1.column_name =  
table2.column_name)] |  
[LEFT|RIGHT|FULL OUTER JOIN table2  
ON (table1.column_name = table2.column_name)];
```



SQL:1999 Compliant Joins

The SQL Compliant joins can obtain the similar results that we have discussed in the previous slides. They differ only in syntax.

The SQL compliant joins are supported from Oracle 9i version onwards

Cross Join

- The Cross Join and Cartesian product are same which produces the cross-product of the tables
- Example: Cross Join on Student_Master and Department_Master

```
SELECT student_name, dept_name  
      FROM student_master  
CROSS JOIN department_master;
```



Cross Join

The example on the slide create a cross product of the two tables. The query result is same as the following query:

```
SELECT student_name,dept_name  
      FROM Student_Master,Department_Master;
```

Natural Join

- The Natural Join is based on the all columns that have same name and datatype in the tables include in the query
- All the rows that have equal values in the matched columns are fetched
- Example: To display student details along with their department details

```
SELECT Student_Code,Student_name,Dept_Code, Dept_name  
      FROM Student_Master  
NATURAL JOIN Department_Master
```



Natural Join

Prior to Oracle 9i, without explicitly specifying the columns of the corresponding tables a join was not possible. With the Natural Join clause, the join can happen automatically based on column names that match in name and datatype.

Oracle returns an error if the datatypes of the columns are different though they have the same name.

The Natural Join is same as EquiJoin.

USING clause

- The USING clause can be replace the NATURAL JOIN if the columns have same names but data types do not match.
- The table name or aliases should not be used in the referenced columns
- This clause should be used to match only one column when there are more than one column matches



USING clause

The NATURAL JOIN returns an error if the datatypes of matching columns are different. In that case the USING clause can be used to specify only those columns on which the join has to done.

The NATURAL JOIN and USING clause are mutually exclusive.

The column used in USING clause cannot be used in WHERE clause

USING clause - Example

- Example 1: To display student details along with their department details. The department code does not match in datatype, hence the join is performed with the USING clause

```
SELECT student_code, student_name, dept_code, dept_name  
      FROM student_master  
      JOIN department_master  
        USING (dept_code, dept_code);
```



ON clause

- Explicit join condition can be specified by using ON clause
- Other search conditions can be specified in addition to join condition
- Example: To display student along with department details from Computer Science department

```
SELECT student.student_code, student.student_name,  
student.dept_code, dept.dept_name  
FROM student_master student  
JOIN department_master dept  
ON (student.dept_Code = dept.dept_Code)  
AND dept.dept_Name ='Computer Science' ;
```



ON clause

With the ON clause you can specify join conditions separate from any other search conditions.

The query is readable and easy to understand

LEFT, RIGHT & FULL Outer Join

- A join between two tables that return rows that match the join condition and also unmatched rows from left table is LEFT OUTER JOIN
- A join between two tables that return rows that match the join condition and unmatched rows from the right table is RIGHT OUTER JOIN
- A join between two tables that return rows that match the join condition and returns unmatched rows of both left and right table is a full outer join



LEFT, RIGHT and FULL OUTER JOIN

The SQL compliant outer join is similar to Oracle proprietary outer join except for the fact that it also has support to perform FULL OUTER JOIN

LEFT, RIGHT & FULL Outer Join - Example

- Example 1: Display student & department details and also those departments who do have students

```
SELECT s.student_code, s.dept_code, d.dept_name  
      FROM student_master s  
RIGHT OUTER JOIN department_master d  
      ON (s.dept_code = d.dept_code);
```

- Example 2 Display student & department details, also those students who are not assigned to any department

```
SELECT s.student_code, s.dept_code, d.dept_name  
      FROM student_master s  
LEFT OUTER JOIN department_master d  
      ON (s.dept_code = d.dept_code);
```



LEFT, RIGHT & FULL Outer Join - Example

- Example 3: Display student & department details. Also those departments who do have students and students who are not assigned to any department

```
SELECT s.student_code,s.dept_code,d.dept_name  
      FROM student_master s  
      FULL OUTER JOIN department_master d  
      ON (s.dept_code = d.dept_code );
```



What is a SubQuery?

- A sub-query is a form of an SQL statement that appears inside another SQL statement.
 - It is also called as a “nested query”.
- The statement, which contains the sub-query, is called the “parent statement”.
- The “parent statement” uses the rows returned by the sub-query.



Sub-queries:

As mentioned earlier, since a basic SQL query returns a relation, it can be used to construct composite queries.

Such a SQL query, which is nested within another higher level query, is called a “sub-query”.

This kind of a nested sub-query is useful in cases, where we need to select rows from tables based on a condition, which depends on the data stored in the table itself.

These can be useful when you need to select rows from a table with a condition that depends on data within the table itself.

Subquery - Examples

- Example 1: To display name of students from “Mechanics” department.

- Method 1:

```
SELECT Dept_Code  
      FROM Department_Master  
     WHERE Dept_name = 'Mechanics';
```

- O/P : 40

```
SELECT student_code,student_name  
      FROM student_master  
     WHERE dept_code=40;
```



Consider the example given on the slide. We want to find details of students from “Mechanics” department. As you can see two queries are used to resolve these problem. Instead of that you can resolve this problem by combining both the queries into one as shown on the next slide.

Subquery - Examples

- Example 1 (contd.):
 - Method 2: Using sub-query

```
SELECT student_code, student_name  
      FROM student_master  
 WHERE dept_code = (SELECT dept_code  
                      FROM department_master  
                     WHERE dept_name = 'Mechanics');
```



The problem is resolved using a one query inside another. The inner query is called as a subquery or nested query. The subquery result is used by the outer query. The subqueries can have more levels of nesting. The innermost query will execute first. The subquery execute only once before the outer query

Where to use Subqueries?

- Subqueries can be used for the following purpose :
 - To insert records in a target table.
 - To create tables and insert records in the table created.
 - To update records in the target table.
 - To create views.
 - To provide values for conditions in the clauses, like WHERE, HAVING, IN, etc., which are used with SELECT, UPDATE and DELETE statements.



- When the WHERE clause needs a set of values which can be only obtained from another query, the Sub-query is used. In the WHERE clause it can become a part of the following predicates
 - COMPARISON Predicate
 - IN Predicate
 - ANY or ALL Predicate
 - EXISTS Predicate.
- It can be also used as a part of the condition in the HAVING clause.

Comparison Operators for Subqueries

- Types of SubQueries
 - Single Row Subquery
 - Multiple Row Subquery.
- Some comparison operators for subqueries:

| Operator | Description |
|----------|---|
| IN | Equals to any member of |
| NOT IN | Not equal to any member of |
| *ANY | compare value to every value returned by sub-query using operator * |
| *ALL | compare value to all values returned by sub-query using operator * |



Sub-queries by using Comparison operators:

Sub-queries are divided as “single row” and “multiple row” sub-queries. While single row comparison operators ($=$, $>$, \geq , $<$, \leq , \neq) can only be used in single row sub-queries, multiple row sub-queries can use IN, ANY or ALL.

For example: The assignment operator ($=$) compares a single value to another single value. In case a value needs to be compared to a list of values, then the IN predicate is used. The IN predicate helps reduce the need to use multiple OR conditions.

The NOT IN predicate is the opposite of the IN predicate. This will select all rows where values do not match the values in the list.

The FOR ALL predicate is evaluated as follows:

- True if the comparison is true for every value of the list of values.
- True if sub-query gives a null set (No values)
- False if the comparison is false for one or more of the list of values generated by the sub-query.

The FOR ANY predicate is evaluated as follows

- True if the comparison is true for one or more values generated by the sub-query.
- False if sub-query gives a null set (No values).
- False if the comparison is false for every value of the list of values generated by the sub-query.

Using Comparison Operators - Examples

- Example 1: To display all staff details of who earn salary least salary

```
SELECT staff_name, staff_code, staff_sal  
FROM staff_master  
WHERE staff_sal = (SELECT MIN(staff_sal)  
                   FROM staff_master);
```

- Example 2: To display staff details who earn salary greater than average salary earned in dept 10

```
SELECT staff_code, staff_sal  
FROM staff_master  
WHERE staff_sal > ANY(SELECT AVG(staff_sal)  
                      FROM staff_master GROUP BY dept_code);
```



For Single Row Subquery the sub-query should result in one value of the same data type as the left-hand side.

Similarly for Multiple Row Subquery the list of values generated by the subquery should be of same data type as the left-hand side

The slide above shows examples of Subqueries. The first query is an example of Single Row Subquery and the second is an example of Multiple row Subquery

Case Study

 Microsoft Word
Document



Copyright © Capgemini 2015. All Rights Reserved 74

Summary

- In this lesson, you have learnt about:
 - DDL commands (CREATE, ALTER and DROP)
 - Constraints
 - Sequence
 - DML command (Insert, Update, Delete)
 - Select Query, Joins and subquery



Review Question

- Question 1: If a sub-query returns multiple values, then the valid operators is/are ____.
 - Option 1: =
 - Option 2: IN
 - Option 3: >
 - Option 4: Any

- Question 2: A sub-query can be used for creating and inserting records.
 - True / False



Review Question

- Question 3 : Inserting rows in a table emp1 from another table can be done using ____.
 - Option 1: insert into emp1(t1) as select empno from emp
 - Option 2: insert into emp1(t1) select empno from emp
 - Option 3: insert into emp1(t1) as select * from emp

- Question 4: Both TRUNCATE statement and DELETE without condition removes the entire date from a table
 - True/False



Review Question

- Question 5 : _____ join is based on any other operator other than equality
- Question 6: _____ join joins the table to itself
- Question 7: _____ join includes a “+” operator along with equality operator



Object-Oriented Programming (OOP) and Unified Modelling Language (UML)

Lesson 05:

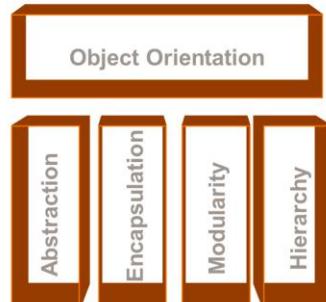
Lesson Objectives

- To understand the following concepts
 - Principles in Object-Oriented technology
 - Abstraction
 - Encapsulation
 - Modularity
 - Hierarchy
 - Polymorphism
 - UML
 - Use Case Diagram
 - Sequence Diagram
 - Class Diagram
 - Demo
 - Case study



Introduction

- OO is based on four basic principles, namely:
- **Principle 1:** Abstraction
- **Principle 2:** Encapsulation
- **Principle 3:** Modularity
- **Principle 4:** Hierarchy



Copyright © Capgemini 2015. All Rights Reserved. 3

Object-Oriented Principles:

Object-Oriented technology is built upon a sound engineering foundation, whose elements are collectively called the “object model”. This encompasses the following principles – Abstraction, Encapsulation, Modularity, and Hierarchy

Each of these principles has been discussed in detail in the subsequent slides.

5.1.1: Object-Oriented Principles

Concept of Abstraction

- Focus only on the essentials, and only on those aspects needed in the given context.
- **For example:** Customer needs to know what is the interest he is earning; and may not need to know how the bank is calculating this interest
- **For example:** Customer Height / Weight not needed for Banking System!



Copyright © Capgemini 2015. All Rights Reserved 4

Abstraction:

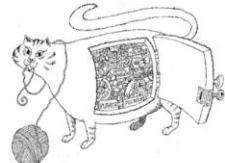
Abstraction is determining the essential qualities. By emphasizing on the important characteristics and ignoring the non-important ones, one can reduce and factor out those details that are not essential, resulting in less complex view of the system. Abstraction means that we look at the external behavior without bothering about internal details. We do not need to become car mechanics to drive a car!

Abstraction is domain and perspective specific. Characteristics that appear essential from one perspective may not appear so from another. Let us try to abstract “Person” as an object. A person has many attributes including height, weight, color of hair or eyes, etc. Now if the system under consideration is a Banking System where the person is a customer, we may not need these details. However, we may need these details for a system that deals with Identification of People.

5.1.2: Object-Oriented Principles

Concept of Encapsulation

- “To Hide” details of structure and implementation
- **For example:** It does not matter what algorithm is implemented internally so that the customer gets to view Account status in Sorted Order of Account Number.



Copyright © Capgemini 2015. All Rights Reserved 5

Encapsulation:

Every object is encapsulated in such a way, that its data and implementations of behaviors are not visible to another object.

Encapsulation allows restriction of access of internal data.

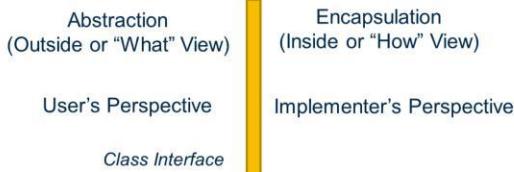
Encapsulation is often referred to as information hiding.

However, although the two terms are often used interchangeably, information hiding is really the result of encapsulation, not a synonym for it.

5.1: Object-Oriented Principles

Encapsulation versus Abstraction

- Abstraction and Encapsulation are closely related.



- Why Abstraction and Encapsulation?
- They result in “Less Complex” views of the System.
- Effective separation of inside and outside views leads to more flexible and maintainable systems.



Copyright © Capgemini 2015. All Rights Reserved 6

Encapsulation versus Abstraction:

The concepts of Abstraction and Encapsulation are closely related. In fact, they can be considered like two sides of a coin. However, both need to go hand in hand. If we consider the boundary of a class interface, abstraction can be considered as the User's perspective, while encapsulation as the Implementer's perspective.

Abstraction focuses on the outside view of an object (i.e., the interface). Encapsulation (information hiding) prevents clients from seeing its inside view, where the behavior of the abstraction is implemented.

The overall benefit of Abstraction and Encapsulation is “Know only that, what is totally mandatory for you to Know”. Having simplified views help in having less complex views, and therefore a better understanding of system. Increased Flexibility and Maintainability comes from keeping the separation of “interface” and “implementation”. Developers can change implementation details without affecting the user's perspective.

5.1: Object-Oriented Principles

Examples: Abstraction and Encapsulation

- Class is an abstraction for a set of objects sharing same structure and behavior



Customer Class

- "Private" Access Modifier ensures encapsulation of data and implementation



Copyright © Capgemini 2015. All Rights Reserved 7

Abstraction and Encapsulation:

When we define a blueprint in terms of a class, we abstract the commonality that we see in objects sharing similar structure and behaviour. Abstraction in terms of a class thus provides the "outside" or the user view.

The implementation details in terms of code written within the operations need not be known to the users of the operations. This is again therefore abstracted for the users. The implementation details are completely encapsulated within the class.

The data members and member functions which are defined as private are "encapsulated" and users of the class would not be able to access them.

5.1.3: Object-Oriented Principles

Concept of Modularity

- Decomposing a system into smaller, more manageable parts
- Example: Banking System can have different modules to take care of Customer Management, Account Transactions, and so on.
- Why Modularity?
 - Divide and Rule! Easier to understand and manage complex systems.
 - Allows independent design and development, as well as reuse of modules.



Copyright © Capgemini 2015. All Rights Reserved 6

Modularity:

Modularity is obtained through decomposition, i.e., breaking up complex entities into manageable pieces. An essential characteristic is that the decomposition should result in modules which can be independent of each other.

As modules are groups of related classes, it is possible to have parallel developments of modules. Changes in one may not affect the other modules. Modularity is an essential characteristic of all complex systems. Well designed modules can be reused in similar situations in other designs.

5.1.3: Object-Oriented Principles

Concept of Modularity

- Modularity in OO Systems is typically achieved with the help of components
- A Component is a group of logically related classes
- A Component is like a black box – users of the component need not know about the internals of a component



Copyright © Capgemini 2015. All Rights Reserved 9

Modularity:

Modularity is one of the corner stones of structured or procedural approach, where functions or procedures are the smallest unit of the application, and they help in achieving the modularity required in the system. In contrast, it is the class which is the smallest unit in OO Systems.

Modularity in OO systems is implemented using Components. A component is a set of logically related classes. For eg. several classes may need to be used together for an application to retrieve data from the underlying databases. So this collection of logically related set of classes for retrieving data can be bundled together as a component for Data Access.

A user of a component need not know about the internals of a component. Modularity thus helps in simplifying the complexity.

5.1.4: Object-Oriented Principles

Concept of Hierarchy

- A ranking or ordering of abstractions on the basis of their complexity and responsibility
- It is of two types:
 - Class Hierarchy: Hierarchy of classes, Is A Relationship.
 - Example: Accounts Hierarchy
 - Object Hierarchy: Containment amongst Objects, Has A Relationship.
 - Example: Window has a Form seeking customer information which has text boxes and various buttons.



Copyright © Capgemini 2015. All Rights Reserved 10

Hierarchy:

Hierarchy is the systematic organization of objects or classes in a specific sequence in accordance to their complexity and responsibility.

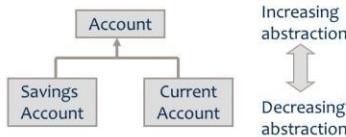
In a class hierarchy, as we go up in the hierarchy, the abstraction increases. So all generic attributes and operations pertaining to an Account are in the Account superclass. Specific properties and methods pertaining to specific accounts like current and savings account is part of the corresponding sub class. Is A relationship holds true – Current Account is an account; Savings Account is an Account.

In object hierarchy, it is the containership property, where one object is contained within another object. So Window contains a Form, a Form contains textboxes and buttons, and so on. Here we have “Has A” relationship – Form has a textbox.

5.1.4: Object-Oriented Principles

Why Inheritance Hierarchy

- Why Inheritance Hierarchy?
 - It is a powerful technique that enables code reuse resulting in increased productivity, and reduced development time.
 - It allows for designing extensible software.

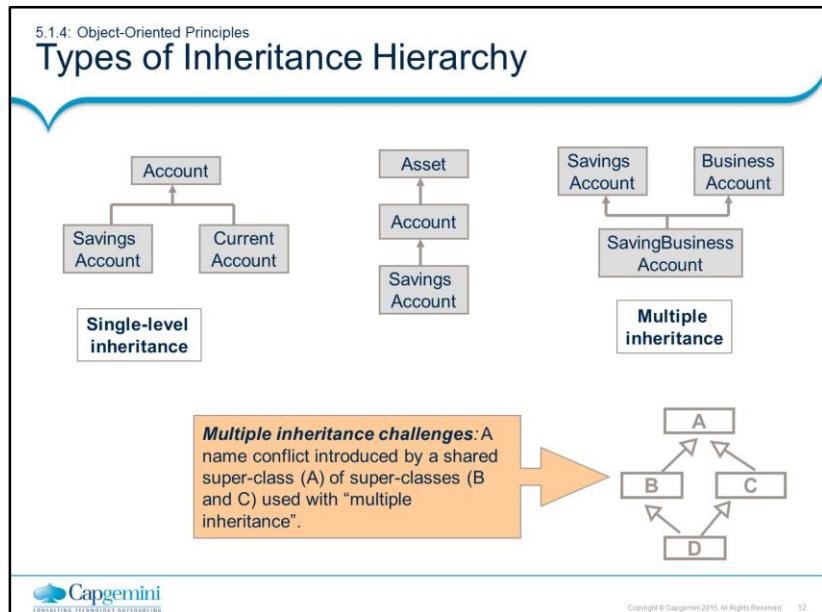


Copyright © Capgemini 2015. All Rights Reserved 11

Why Inheritance Hierarchy?

Inheritance is the process of creating new classes, called derived classes, from existing or base classes. The derived class inherits all the capabilities of the base class, but can add some specificity of its own. The base class is unchanged by this process.

Once the base class is written and debugged, it need not be touched again, but can nevertheless be adapted to work in different situations. Reusing existing code saves time and money and increases the program reliability.



Types of Inheritance Hierarchy:

Single-level inheritance: It is when a sub-class is derived simply from its parent class.

Multilevel Inheritance: It is when a sub-class is derived from a derived class. Here a class inherits from more than one immediate super-class. Multilevel inheritance can go up to any number of levels.

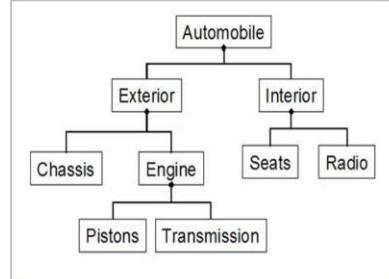
Multiple Inheritance: It refers to a feature of some OOP languages in which a class can inherit behaviors and features from more than one super-class.

Finally, we could have Hybrid inheritance, which is essentially combination of the various types of inheritance mentioned above.

5.1.4: Object-Oriented Principles

Object Hierarchy

- “Has-a” hierarchy is a relationship where one object “belongs” to (is a part or member of) another object, and behaves according to the rules of ownership.



Copyright © Capgemini 2015. All Rights Reserved 13

Note: The container hierarchy (has-a hierarchy) is in contrast to the inheritance hierarchy, i.e., the “generic-specific” levels do not come in here.

5.1.4: Object-Oriented Principles

A glance at relationships

- The Inheritance or “Is A” Hierarchy leads to Generalization relationship amongst the classes.
- The Object Hierarchy or “Has A” relationship leads to Containment relationship amongst the objects
- Aggregation and Composition are two forms of containment amongst objects
- Aggregation is a loosely bound containment. Eg. Library and Books, Department and Employees
- Composition is tightly bound containment. Eg. Book and Pages



Copyright © Capgemini 2015. All Rights Reserved 14

As seen earlier, in an inheritance hierarchy, the super class is the more generic class, and subclasses extend from the generic class to add their specific structure and behaviour. The relationship amongst these classes is a generalization relationship. OO Languages provide specific syntaxes to implement inheritance or the generalization relationship. Has A or Containment is further of two forms depending on how tight is the binding between the container (“Whole”) and its constituents (“Part”). In the whole-part relationship, if the binding is loose i.e. the contained object can have an independent existence, the objects are said to be in an aggregation relationship. On the other hand, if the constituent and the container are tightly bound (Eg. Body & parts like Heart, Brain..), the objects are said to be in a composition relationship.

5.1.4: Object-Oriented Principles

A glance at relationships

- Most commonly found relationship between classes is Association
- Association is the simplest relationship between two classes
- Association implies that an object of one class can access public members of an object of the other class to which it is associated



Copyright © Capgemini 2015. All Rights Reserved 15

The relationship we are most likely to see amongst classes is the Association Relationship. When two classes have an association relationship between them, it would mean that an object of one class can access the public members of the other class with which it is associated. For eg. if a “Sportsman” class is associated with “Charity” class, it means that a Sportsman object can access features such as “View upcoming Charity Events” or “Donate Funds” which are defined within the “Charity” class.

5.2: Polymorphism

Key Feature – Polymorphism

- It implies One Name, Many Forms.
- It is the ability to hide multiple implementations behind a single interface.
- There are two types of Polymorphism, namely:
 - Static Polymorphism
 - Dynamic Polymorphism



Copyright © Capgemini 2015. All Rights Reserved 16

Polymorphism:

The word Polymorphism is derived from the Greek word “Polymorphous”, which literally means “having many forms”. Polymorphism allows different objects to respond to the same message in different ways!

There are two types of polymorphism, namely:

- Static (or compile time) polymorphism, and
- Dynamic (or run time) polymorphism

5.2: Polymorphism

Key Feature – Static Polymorphism

- Resolution of the “Form” is at compile time, achieved through overloading.

```
int addInteger(int, int);  
float addFloat(float, float);  
double addDouble(double, double);
```

Traditional Approach

Overloaded Functions

```
int addNumber(int, int);  
float addNumber(float, float);  
double addNumber(double, double);
```

Resolution
At
Compile
Time

Though the name is the same, the right function is called depending on number and/or types of parameters that are there in the function invocation



Copyright © Capgemini 2015. All Rights Reserved 17

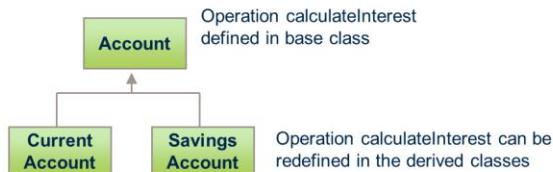
Polymorphism (contd.):

Overloading is when functions having same name but different parameters (types or number of parameters) are written in the code. When Multiple Sort operations are written, each having different parameter types, the right function is called based on the parameter type used to invoke the operation in the code. This can be resolved at compile time itself since the type of parameter is known.

5.2: Polymorphism

Key Feature – Dynamic Polymorphism

- Resolution of the “Form” is at run time, achieved through overriding.



The right operation defined in one of these classes is invoked at Run Time depending on which object is invoking the operation.



Copyright © Capgemini 2015. All Rights Reserved 18

Polymorphism (contd.):

On the other hand, the function calculateInterest can be coded across different account classes. At runtime, based on which type of Account object (i.e., object of Current or Savings Account) is invoking the operation, the right operation will be referenced. Overriding is when functions with same signature provide for different implementations across a hierarchy of classes.

As seen in the example here, inheritance hierarchy is required for these objects to exhibit polymorphic behaviour. The classes here are related since they are different types of Accounts, so it is possible to put them together in an inheritance hierarchy. Does that mean that polymorphism is possible only with related classes in an inheritance hierarchy? The answer is No!

We can have unrelated classes participating in polymorphic behavior with the help of the “Interface” concept, which we shall study in a subsequent section.

5.2: Polymorphism

Key Feature – Polymorphism

Why Polymorphism?

- It provides flexibility in extending the application.
- It results in more compact designs and code.



Copyright © Capgemini 2015. All Rights Reserved 19

Polymorphism (contd.):

If the banking system needs a new kind of Account, extending without rewriting the original code, then it is possible with the help of polymorphism.

In a Non OO system, we would write code that may look something like this:

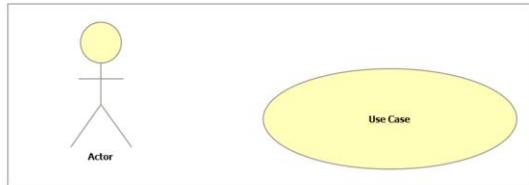
```
IF Account is of CurrentAccountType THEN  
    calculateInterest_CurrentAccount()  
IF Account is of SavingsAccountType THEN  
    calculateInterest_SavingAccount()
```

The same in a OO system would be myAccount.calculateInterest(). With object technology, each Account is represented by a class, and each class will know how to calculate interest for its type. The requesting object simply needs to ask the specific object (For example: SavingsAccount) to calculate interest. The requesting object does not need to keep track of three different operation signatures.

5.3.1: Use Case Diagrams

Use Case Diagrams - Features

- Use Case Diagrams model the functionality of a system by
- using Actors and Use Cases:
 - Actor is a user of the system.
 - Use cases are services or functions provided by the system to its users.



Copyright © Capgemini 2015. All Rights Reserved 20

Use Case Diagrams:

Use Case is a description of a system's behavior from a user's point of view. It is a set of scenarios that describe an interaction between a user and a system. It also displays the relationship among Actors and Use Cases. Two main components of Use Case diagram are Use Cases and Actors.

Use case diagrams, which render the User View of the system, describe the functionality (Use Cases) provided by the system to its users (Actors).

An Actor represents a user or another system that will interact with the system you are modeling.

An Use Case is an external view of a system that represents some action that the user might perform in order to complete a task.

5.3.1: Use Case Diagrams

Definition of Actor

- Actor:

- An Actor can be defined as follows:
 - Actor is any entity that is external to the system and directly interacts with the system, thus deriving some benefit from the interaction.
 - Actor can be a human being, a machine, or a software.
 - Actor is a role that a particular user plays while interacting with the system.
 - Examples of Actors are End-user (roles), External systems, and External passive objects (entities).



Copyright © Capgemini 2015. All Rights Reserved 21

Actor:

Actors are people, organizations, systems, or devices which use or interact with our system. The system exists to support that interaction. Therefore, the important part of the project is to identify the Actors and find out what they want from the system.

Actors are characterized by their external view rather than their internal structures. It is a role that the user plays to get something from the system.

Role and organization Actors only require logical interactions with the system. Ask who wants what from our system, rather than who operates the system.

For example: ABC and XYZ are users who wish to buy from an online store. For the online stores system, they play the role of a customer, and hence customer is the Actor for the system. The database for this system may already be existing, and hence this may be another Actor (note that user in this case is not a human).

The Actors will finally be used to describe classes, which will interact with other classes of the system.

5.3.1: Use Case Diagrams

Definition of Use Cases

- Use Case:
 - An Use Case can be defined as a set of activities performed within a system by a User.
 - Each Use Case:
 - describes one logical interaction between the Actor and the system.
 - defines what has changed by the interaction.



Copyright © Capgemini 2015. All Rights Reserved 22

Use Cases:

The Use Cases define “units of functionality” provided by system. They model “work units” that the system provides to its outside world.

A Use Case is one usage of the system. It is a generic description of a use of the system. It allows interactions in a specific sequence.

At the lowest level, they are nothing but methods which need to be implemented by various classes in the system.

Use Cases determines everything that the Actor wants to do with the system.

A Use Case performs the following functions:

Defines main tasks of the system

Reads, writes, and changes system information

Informs the system of real world changes

A Use Case needs to be updated / informed about system changes.

5.3.1: Use Case Diagrams

Drawing the Use Case Diagram

- A Use Case diagram has the following elements:
- **Stick figure:** It represents an Actor.
- **Oval:** It represents a Use Case.
- **Association lines:** It represents communication between Actors and Use Cases.



Copyright © Capgemini 2015. All Rights Reserved. 23

Drawing the Use Case Diagram:

The Use Case Diagram has the following elements:

A stick figure, which represents Actors

(sometimes stereotyped classes, as explained later, are also used to represent Actors). They differ from tool to tool.

Ovals or ellipses, which represent Use Cases

Association lines, which indicate interactions between Actors and Use Cases.

Use Cases will have description of what the Use Case is supposed to do when it is used.

An example of use case description is given.

5.3.1: Use Case Diagrams

Use Case Relationships - Overview

- Types of relationships between Use Cases are:
 - Include
 - Extend



Copyright © Capgemini 2015. All Rights Reserved 24

Use Case Relationships:

Relationships help us connect the model elements.

After finding out the primary Use Cases, one can start looking “into” the system to see if there are any relationships between the Use Cases.

The following types of relationships can exist between the Use Cases:

include
extend

5.3.1: Use Case Diagrams

Include relationship - Characteristics

- Include relationship:
 - «include» stereotype indicates that one use case “includes” the contents of another use case.
 - Include relationship enables factoring out frequent, common behavior.
- Use case “A” includes use case “B”, if:
 - B describes scenario which is part of scenario of A, and
 - B describes scenario common for a set of Use Cases including A.



Copyright © Capgemini 2015. All Rights Reserved 25

Use Case Relationship – Include:

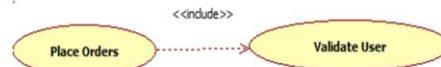
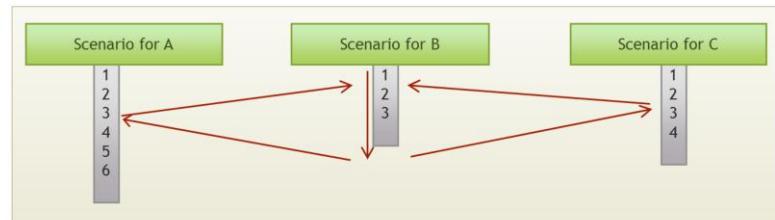
In an Include relationship, one Use Case includes behavior specified by another Use Case. If there are common steps in the scenarios of many Use Cases, they can be factored out into a separate Use Case. This Use Case can then be included as part of the “Primary Use Case”.

The above arrangement helps us segregate and organize common sub-tasks. An “Included Use case” is not a complete process. Extra behavior is added to the base Use Case.

This may also be used in case of complex Use Cases (where there is too much functionality in one Use Case). In such cases, the primary functionality can be distributed across Use Cases, and a primary Use Case can then include the remaining secondary Use Cases.

5.3.1: Use Case Diagrams

Include relationship - Example



Copyright © Capgemini 2015. All Rights Reserved 26

Use Case Relationships – Include (contd.):

As illustrated in the figure shown in the slide, scenario of Use Case B is required by Use Case A and Use Case C. Hence both Use Cases A and C can include Use Case B. After completing the scenario of Included Use Case B, the Use Cases A and C will continue with their respective scenarios.

5.3.1 : Use Case Diagrams

Extend relationship - Characteristics

- Extend relationship:
 - «extend» stereotype indicates that one Use Case is “extended” by another Use Case.
 - Extend relationship enables factoring out infrequent behavior or error conditions.
 - Extend relationship represents optional behavior for a Use Case which will be required only under certain conditions.



Copyright © Capgemini 2015. All Rights Reserved 27

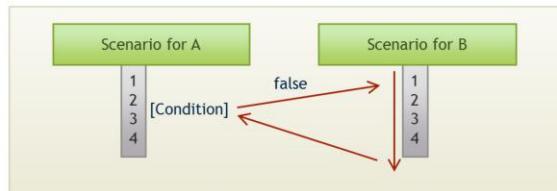
Use Case Relationships - Extend :

In an Extend relationship, an Use Case may be required by another use case based on “some condition”, or due to “some exceptional situation”.

Again, upon completion of extension activity sequence, the original Use Case will continue.

5.3.1 : Use Case Diagrams

Extend relationship - Example



Copyright © Capgemini 2015. All Rights Reserved 28

Use Case Relationships – Exclude (contd.):

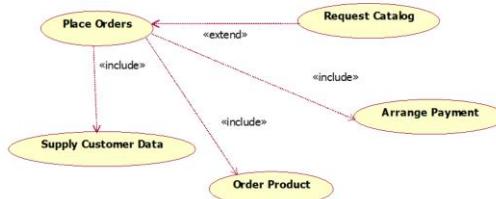
As illustrated in the figure shown in the slide, in Use Case A when the condition becomes false, the scenario of Use Case B is invoked.

Note that Use Case B is said to extend Use Case A. The stereotyped generalization arrow with keyword “extend” depicts the extend relationship between the Use Cases.

5.3.1 : Use Case Diagrams

Examples of Use Case Relationships

- Example 1:



Copyright © Capgemini 2015. All Rights Reserved 29

Example of Use Case Diagram:

The slide shows an example where we are looking at the Use Case relationships (though the Actors and system boundary has not been shown here, let us assume that they exist. They have been left out so as to focus on the relationships).

The Request for Catalog may not always happen when an Order needs to be placed. Hence, Extend is the relationship used between the two Use Cases of “Place Order” and “Request Catalog”.

“Place Order” being a complex Use Case, it is broken down into secondary use cases of:

Supply Customer data

Order Product

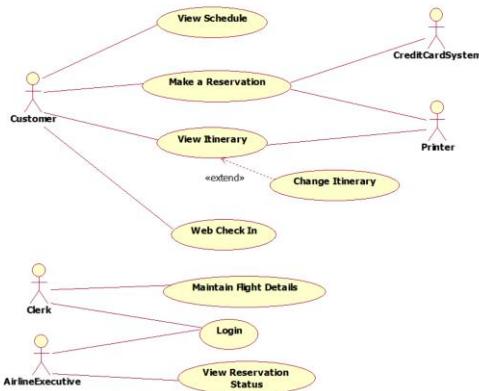
Arrange Payment

It is important to note that the diagram by itself does not indicate any kind of ordering (i.e. first invoke order product, and then arrange payment, etc). The ordering has to be taken care of as part of the Use Case scenario.

5.3.1 : Use Case Diagrams

Examples of Use Case Relationships (Contd...)

Example 2:



Copyright © Capgemini 2015. All Rights Reserved 30

How do you interpret this diagram?

5.3.2 : Sequence Diagrams

Features

- **Sequence Diagram:**
 - A Sequence diagram describes interactions among classes in terms of an “exchange of messages over time”.
 - Some of the facts related to Sequence Diagrams are:
 - Sequence Diagrams are used to depict the time sequence of messages exchanged between objects.
 - Messages can correspond to operation on class or a event trigger.



Copyright © Capgemini 2015. All Rights Reserved 31

Sequence Diagrams: Features:

Starting from the scenarios of the Use Case, interactions between objects in the scenario can be depicted in the Sequence Diagram. These interactions are in the “sequence of time”, and finally correspond to “operations” or “event triggers”.

Only the sequences of messages, with respect to time, are important. There is no particular span of time that is considered.

Sequence Diagrams show objects, and not classes. This is different from Class Diagrams, which contains classes that signify the existence of objects.

Sequence Diagrams provide a better correlation between the “dynamic view” of the system and the “static view” held in the Class Diagram.

5.3.2 : Sequence Diagrams

Notations

- Notations of a Sequence Diagram include:
- **LifeLine:** It is a vertical dashed line that represents the “lifetime” of an object.
- **Arrows:** They indicate flow of messages between objects.
- **Activation:** It is a thin rectangle showing period of time, during which an object is performing an action.



Copyright © Capgemini 2015. All Rights Reserved 32

Sequence Diagrams: Notations:

In the Sequence Diagram, different objects / class roles are laid out horizontally at the top. There is no significance to the ordering of these objects.

The Lifeline, denoted as dashed line beneath the class role, is used to model “existence of entities over time”.

Activation, shown as thin rectangles along the lifeline, model the time during which entities are active. Activation may not always be depicted.

Arrows indicate flow of messages between objects.

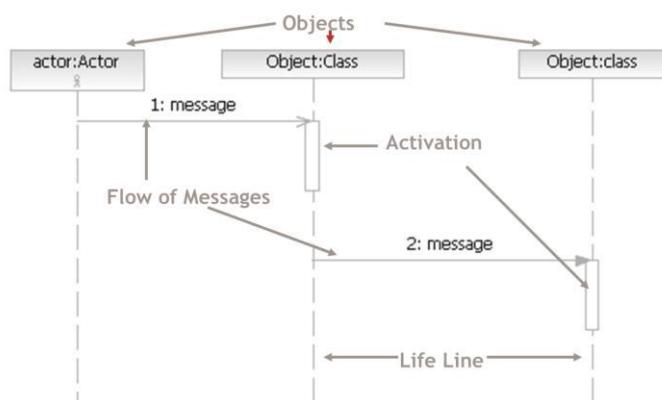
Messages are denoted as horizontal arrows between lifelines.

Messages may be:

as simple as a “string” conveying an operation, or as detailed as a “method”, which includes parameters passed and values returned.

5.3.2 : Sequence Diagrams

Notations (Contd...)



The slide shows the notations used in a Sequence Diagram.

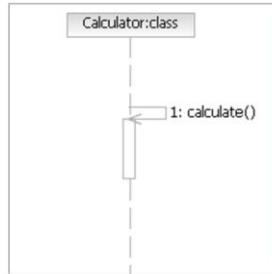
5.3.2 : Sequence Diagrams

Direction of Arrows

- Direction of Arrows:

- Direction indicates which object's method is being called by whom.

- A circulating arrow on the Object Lifeline is for a self method - called within the object by itself.



Copyright © Capgemini 2015. All Rights Reserved 34

Sequence Diagram: Notations (Directions and Branches):

Direction of arrows:

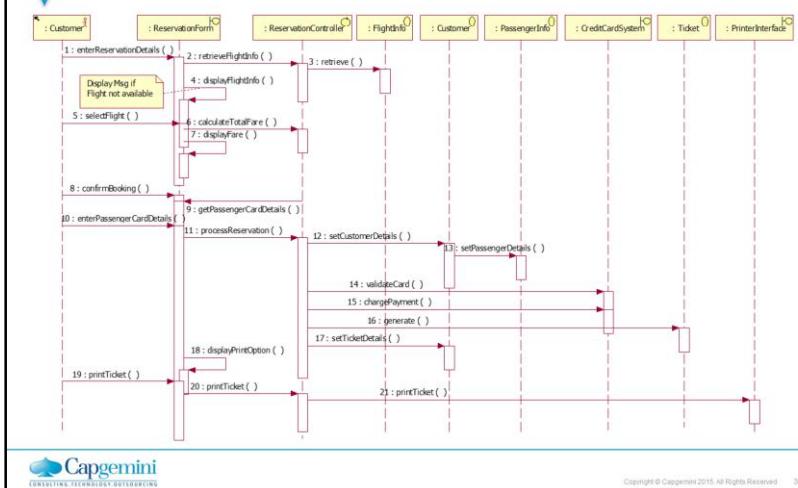
Every message has a “sender” and “receiver” and this is depicted by the direction of the arrow head. Control gets passed from the sender to receiver.

“Implicit returns” are assumed in case there are no return messages to the sender.

It is possible that there is a “call to object’s own method”. A “circulating arrow” is used to depict such a case. In the example shown on the slide, Calculator object calls its own method calculate().

5.3.2 : Sequence Diagrams

Example of Sequence Diagrams



How do you interpret this diagram?

Copyright © Capgemini 2015. All Rights Reserved. 35

5.3.3: Class Diagrams

Features

- **Class Diagrams:**
- Class Diagrams define the basic building blocks of a model, namely:
 - types
 - classes, and
 - general material used to construct the full model



Copyright © Capgemini 2015. All Rights Reserved. 36

Class Diagrams: Features:

Class Diagrams can be used to model classes, and the relationships between classes.

When drawn during the analysis stage, only the names of the classes maybe represented.

During further refinements in the detailed analysis or design stage, details like “attributes” and “services” get added to each class, and are depicted in the Class Diagram.

5.3.3 : Class Diagrams

Functions

- Class Diagrams have the following functions:
 - They describe the static structure of a system.
 - They show the existence of classes and their relationships.
 - Classes represent an abstraction of entities with common characteristics.
 - Relationships may be:
 - Generalization
 - Association
 - Aggregation
 - Composition, or
 - Dependency



Copyright © Capgemini 2015. All Rights Reserved 37

5.3.3 : Class Diagrams

Uses

- Typical uses of Class Diagrams are:
- To model vocabulary of the system, in terms of system's abstractions
- To model collaborations between classes
- To model logical database schema (blueprint for conceptual design of database)



Copyright © Capgemini 2015. All Rights Reserved 38

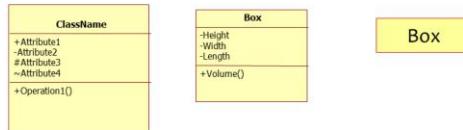
Uses of Class Diagram:

The importance of the Class diagram is that it gives a view of all the classes that are required to make up the system. It also conveys the collaborations that exist between classes to give the system behavior.

5.3.3 : Class Diagrams

Notations for Class

- Class may be represented in any of the following ways
- Only Class Name is mandatory



Notations for Class:

Classes are denoted as rectangles, with compartments for name, attributes, and operations. There is optionally a last compartment that can be used for specifying responsibilities, variations, business rules, etc.

The name is the mandatory part. Other compartments may be included based on the amount of details required to be communicated. The representations of classes that do not have all compartments are known as “elided notations for class”.

5.3.3 : Class Diagrams

Notations for Class (Contd...)

- Class Visibility signifies how information within class can be accessed.

| Symbol | Meaning |
|--------|-----------|
| + | Public |
| - | Private |
| # | Protected |



Copyright © Capgemini 2015. All Rights Reserved 40

Notations for Class: Class Visibility:

Information about visibility of attributes and operations can sometimes be represented by using symbols like + for public, - for private, or # for protected.

These symbols may vary from tool to tool.

5.3.3 : Class Diagrams

Association Relationship - Features

- In Association:
 - Name indicates relationship between classes.
 - Role represents the way classes see each other.



Copyright © Capgemini 2015. All Rights Reserved. 41

Relationships: Association:

Associations may be characterized by the following:

Name: The name signifies purpose of association, and is written along with the line indicating association, role and direction of association.

Role: In case there are specific roles played by classes in the association, then it is indicated by the role name, which is written near the class.

Arrow: Arrows may be used to indicate whether the association is uni-directional or bi-directional. Absence of arrows implies that no inferences can be drawn about the navigability.

The example in the slide shows an association relationship between a class Person and a class Car. The class Person plays the role of an owner.

5.3.3 : Class Diagrams

Association Relationship - Example

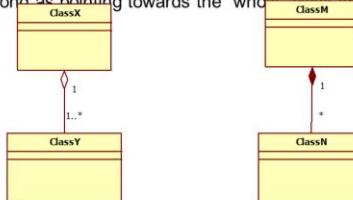


5.3.3 : Class Diagrams

Relationships - Features

- Aggregation and Composition:

- The following Class Diagram, possessing Composition and Aggregation, displays:
 - Aggregation as indicated by a hollow diamond.
 - Composition as indicated by a filled diamond.
 - Diamond as pointing towards the “who” part of the aggregate.



Relationships: Aggregation and Composition:

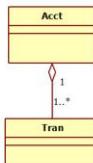
Composition and Aggregation are modeled with filled diamond and hollow diamond, respectively, on the “Whole” part.

Roles and multiplicity, if required, can be mentioned here, as well. Typically they are done for the “Part” part of the “Whole” part.

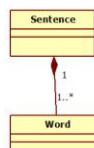
5.3.3 : Class Diagrams

Relationships - Examples

Aggregation



Composition



Examples of Aggregation and Composition:

In the examples shown in the slide,

the relationship between a Sentence and a Word is represented as a “Composition” (Word is a part of a sentence).

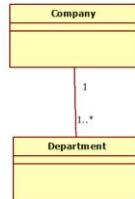
the relationship between Account and Transaction is represented as an “Aggregation”.

5.3.3 Class Diagrams

Definition of Multiplicity

- **Multiplicity:**
- Multiplicity indicates the “number of instances” of one class linked to “one instance” of another class.

| Symbol | Meaning |
|--------|--------------|
| 1 | Exactly one |
| 0..1 | Zero or one |
| * | Many |
| 0..* | Zero to many |
| 1..* | One to many |



Copyright © Capgemini 2015. All Rights Reserved. 45

Multiplicity:

Multiplicity attached to a class denotes the possible cardinalities of objects of the association.

For example: The above figure depicts that “One company has one or more departments, and a department is associated with one company”.

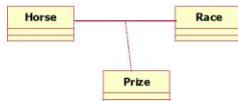
Multiplicity values can be indicated in association, aggregation, and composition relationships.

5.3.3 : Class Diagrams

Association Class Relationship - Features

- Association Class:

- An Association Class is a class that has properties of both an “association” and a “class”.
- It is required when properties result from unique combination of two classes.
- For example:



Copyright © Capgemini 2015. All Rights Reserved. 46

Association class:

An Association class is a class required as the result of association between two classes.

For example:

The Prize class is a result of association between the Horse class and the Race class.

For each Horse placed in a Race there is a prize.

The amount of prize depends on the race.

The Prize class could not be associated with the Horse class alone because a Horse might have many Prizes, and the relationship between the Prize and Race would be lost.

Similarly, Prize class cannot be associated with Race class alone because a Race has many Prizes, and the relationship between the Prize and the Horse would be lost.

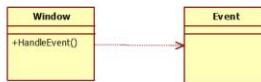
Similarly, result of a student (in terms of marks in assignments, test, and grade) in a course is a unique combination of an individual student, and a particular course. So we can have an association between Student and Course Classes, with Result being an Association Class.

5.3.3 : Class Diagrams

Dependency - Features

- **Dependency:**

- Dependency is a "using" relationship within which the change in the specification of one class may affect another class that uses it.
- For example:



Copyright © Capgemini 2015. All Rights Reserved. 47

Dependency:

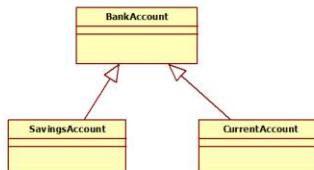
The dependencies are denoted as dashed arrows with arrow head pointing to the independent element.

In the example shown in the slide, the structure and behavior of the Window Class is dependent on the structure and behavior of the Event Class.

5.3.3 : Class Diagrams

Generalization - Features

- Generalization:
 - Generalization indicates relationships between super-class and sub-class.
 - For example:

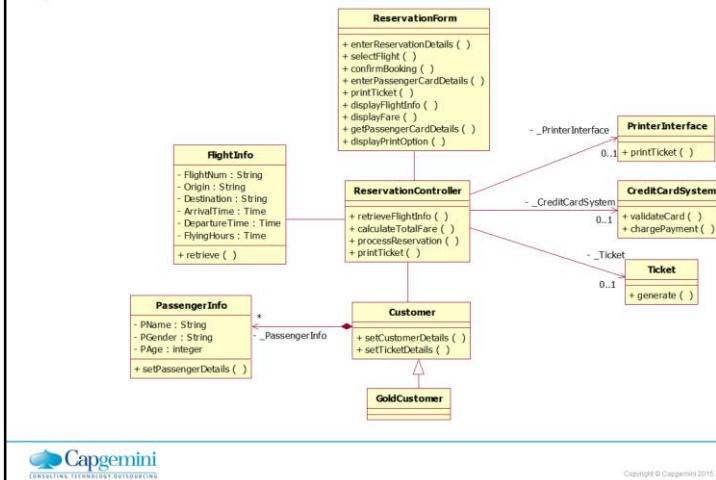


Relationships: Generalization:

Generalizations are denoted as “paths” from specific elements to generic elements, with a hollow triangle pointing to the more general elements.

5.3.3 : Class Diagrams

Example of Class Diagrams



How do you interpret this diagram?

5.3.4 Demos

Demo



- Demo on
 - Use case diagram
 - Sequence diagram
 - Class diagram



 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 50

5.3.5 Case Study

Scenario from Banking System

- Geetha and Mahesh hold accounts in Bank XYZ Ltd. Geetha has a savings as well as a current account with the bank. Mahesh only has a current account. As customers of the bank, Geetha and Mahesh can deposit or withdraw money from their accounts as per the norms and policies defined by the bank on savings and current accounts.
- Bank XYZ Ltd. continuously adds new customers to its existing customer base. Of course, some its customers may also want to close their accounts due to changing needs of the customer.



5.3.5 Case Study

Scenario from Banking System

- Identify classes, attributes and methods
- Draw an use case diagram to identify the actors and their functionalities they perform.
- Draw the class diagram and identify correct relationship between the classes



Copyright © Capgemini 2015. All Rights Reserved 52

Summary

- In this lesson, you have learnt about:
 - Principles in Object-Oriented technology
 - Abstraction
 - Encapsulation
 - Modularity
 - Hierarchy
 - Polymorphism
 - UML diagram
 - Use Case Diagram
 - Sequence Diagram
 - Class Diagram
 - Demo
 - Case study



Review Question

- Question 1: The 4 basic principles of Object Model are ___, ___, ___ and ___.
- Question 2: Function Overriding is kind of polymorphism.
 - True / False
- Question 3: ___ hierarchy is a relationship where one object behaves according to the rules of ownership.



Review Question

- Question 4: Abstraction focuses on:
 - Option 1: implementation
 - Option 2: observable behavior
 - Option 3: object interface

- Question 5: Polymorphism can be achieved by:
 - Option 1: Hierarchy of Classes providing polymorphic behavior
 - Option 2: Interfaces
 - Option 3: Containment of Objects

- Question 6: A message in a Sequence Diagram will help identifying ____.



Review Question

- Question 7: Use Case Diagrams represent the functionality needed in a system.
 - True / False

- Question 8: A Class Diagram gives information about:
 - A. Attributed defined for a class
 - B. Operations defined for a class
 - C. Logic to be used for an operation of a class

- Question 9: Relationships that you may find on a Class Diagram are ___, ___, ___, ___ and ___.



Software Engineering

Lesson 06:

Lesson Objectives

- To Understand the following :
 - Different Phases in Software Development Life Cycle
 - Requirements Phase
 - Design Phase
 - Construction Phase
 - Testing Phase
 - Acceptance Phase
 - Review Process
 - Configuration Management Process



6.1 SDLC

Software Development Life Cycle (SDLC)

- Also known as software development process or Systems development life cycle
- A set of processes, standards and tools used to develop, alter software in a optimal manner
- Starts when a product is conceived and ends when the product is no longer available or is effective to use
- Composed of phases , where each phase is dependent on the previous phase's result
- Each phase is a limited period of time starting with a definite set of data and having a definite set of results

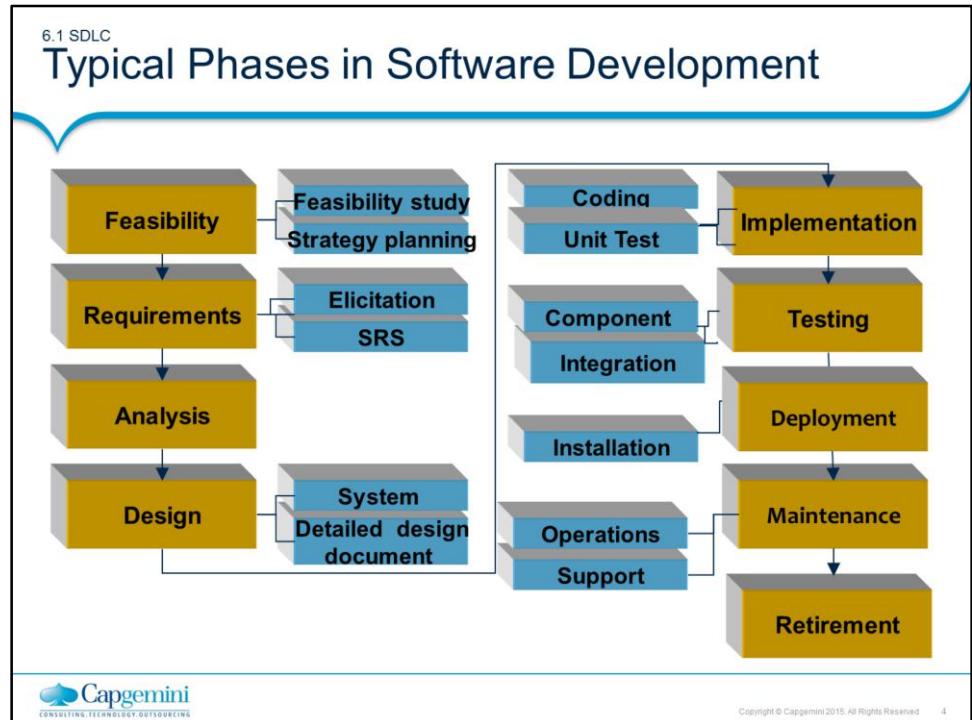


Copyright © Capgemini 2015. All Rights Reserved 3

Also known as **systems development life cycle (SDLC)**, or **software development process**, or **Software Development Life Cycle**

It is a process of creating or altering information systems using various models and methodologies. The SDLC aims to produce a high quality system that meets or exceeds customer expectations, reaches completion within times and cost estimates, works effectively and efficiently.

The SDLC framework provides a sequence of activities for system design and development . It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one.



Activities during phases

Requirements: establish the customer's needs

System Design: develop the system's structure

Detailed Design: develop module structures

Implementation: code or otherwise

Testing: check what's been developed

Installation: bring the system into production

Maintenance: correct, adapt, improve

6.1.1 Requirements Phase

What is a Requirement?

- Simply put , it is the needs of the stakeholder which needs to be met/satisfied (by a s/w)
- A Software capability needed by the User to solve a problem to achieve an objective.
- Can be a high level abstract statement indicating needs to a details of the system which the client can validate
- Requirements needs to be
 - Elicited
 - Analyzed
 - Specified
 - Managed
- The engineering process covering all activities leading to discovery , document and manage requirement is known as Requirement Engineering



Copyright © Capgemini 2015. All Rights Reserved 5

A requirement is a capability or condition to which the system must conform. Software requirements provide a “black box” definition of the system. They define only those externally observable “What’s” of the system, not the “How’s.”

Requirements are very important for any project, or sub-section of a project, because they define what will be built, hence requires a rigorous engineering process, , hence the term Requirement engineering .

Requirement engineering is a continuous activity throughout the lifetime of a software as requirements are subject to change . New requirements needs to be elucidated existing requirements revamped etc.

6.1.1 Requirements Phase

Requirement phase

- This is the initial phase of the development process
- The development team works closely with the customer to determine the customer's requirements for the product – functional, non functional and other characteristics which the product must mandatorily have .
- The requirements identified in this phase serve as a foundation for the remaining phases of the development process, and the customer acceptance criteria.
- The main participants involved in the requirement phase are
 - Stake holders
 - Requirement Engineer



Copyright © Capgemini 2015. All Rights Reserved 6

Stakeholders are individuals who affect or are affected by the software product
 They have some influence over the software , in terms of requirements
 Stakeholders can be categorized as

- Acquirers of the software (both management and users)
- Suppliers of the software (individuals and team , management)
- Others (Sales , Legal teams , other internal teams)

RE who are also known as requirements engineer, business analyst, system analyst, product manager, or simply analyst

RA's primary responsibility is to gather, analyze, document and validate the needs of the project stakeholders. They help to determine the difference between what customers say they want and what they really need

Identifying and considering the needs of all of the different stakeholders can help prevent requirements from being overlooked.

Requirements can be classified under two categories :

Functional : Requirements what the system should do or provide for users .They can include all the business processes /functionality, reports and queries and details of data to be stored and managed .

Non Functional : Non-functional requirements are constraints, targets or control mechanisms for the new system. They describe how, how well the system should provide services like response time , ease of use-usability , security, recoverability etc.

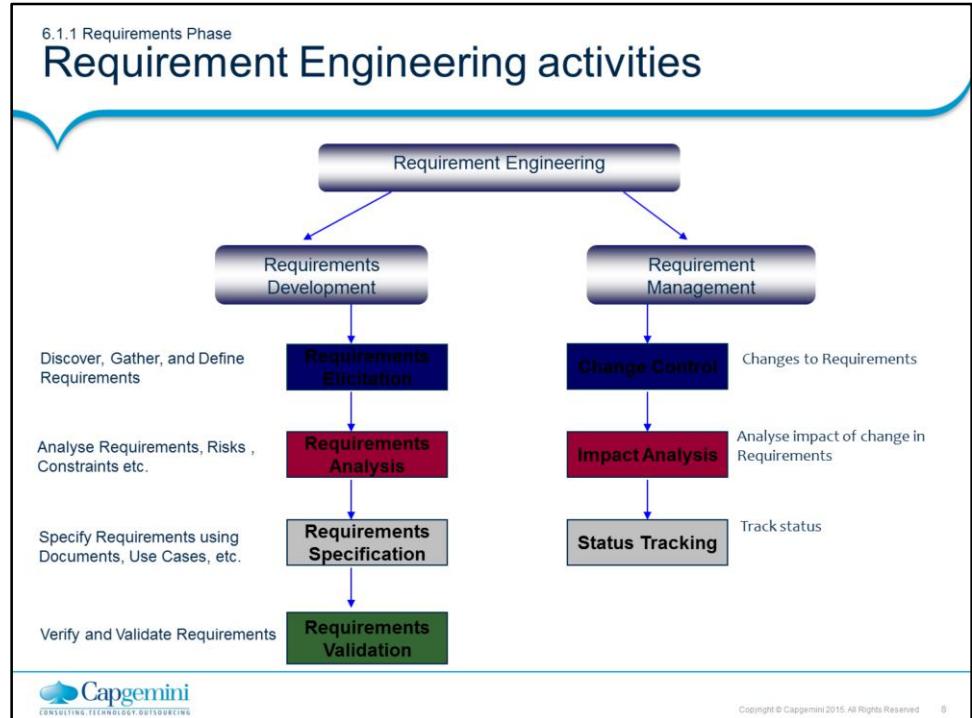
6.1.1 Requirements Phase

Need for good requirements

- Requirement Problems are the single No.1 reason for projects failing over
 - Schedule
 - Budget
 - Scope
 - Quality
 - And even getting Cancelled!!
- Reworking requirements cost 40-50% of project effort
- Many problems found during design, testing, or operation of a system are the result of incorrect, incomplete, or missing requirements



Copyright © Capgemini 2015. All Rights Reserved 7



Requirements Engineering = Requirements Development + Requirements Management

6.1.1 Requirements Phase

Requirement Engineering activities

▪ Requirement Elicitation

- This phase focuses on examining and gathering desired requirements and objectives for the system from different stakeholders
- Various techniques are followed to gather requirements viz interviews, document examining , brainstorming , prototyping etc

▪ Requirement Analysis

- This phase focusses on analyzing , rigorously , classifying, prioritizing , documenting the gathered requirements within business context

▪ Requirement Specification and Validation

- A formal document is prepared after collating all requirements which contains a complete description of the external behavior of the software system.
- Requirements are specified in
 - URS User Requirement Specification
 - SRS System Requirement Specification
 - Use Case Documentation
- The requirement documented in the SRS is verified , validated and agreed upon by all parties .



Copyright © Capgemini 2015. All Rights Reserved 9

Requirements Specifications include

What is in scope and out of scope

Related or referenced documents (Customer supplied artifacts and materials)

Requirement providers and stakeholders of the project

Deliverables & delivery dates

Risks and assumptions

Current and proposed business system

Acceptance criteria and Customer CTQs

Functional and non functional requirements

Limitations and constraints

URS : User Requirement Specification

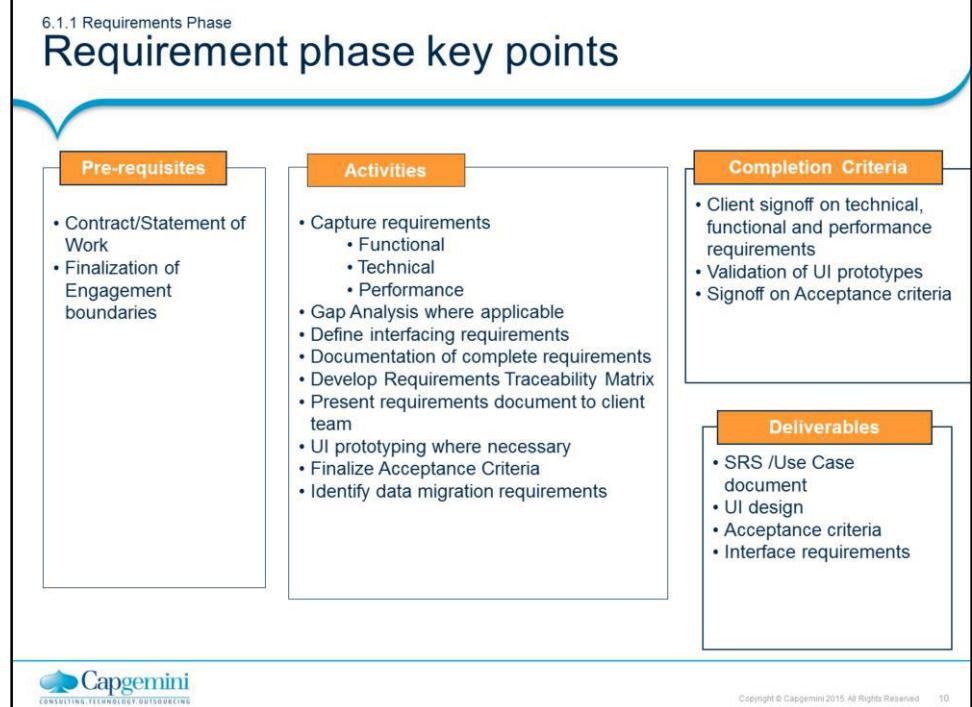
Typically written prior to the SRS, based on the user's experience and expectations, with inputs from stakeholders

SRS : System Requirement Specification

This information includes detailed descriptions of the operations performed by each screen, the data that can be entered into the system , work-flows performed by the system and system reports or other outputs, . An SRS also specifies who can enter data into the system as well as how the system meets regulatory requirements that are applicable to the specific system.

Use Case Documents : The document and diagrams together forms the UCD . Typically done when the approach is Use case modelling

QMS provides templates for creating specification document



6.1.2 Design Phase

Architecture and Design

▪ Architecture

- It is the high level organizing structure of the system
- It defines the components, interfaces, and behaviors of the system.
- The process of architecting a software involves defining a structured solution that meets all of the technical and operational requirements, along with attributes such as performance, security, and manageability.
- This phase usually involves the technical/solution architect

▪ Design

- It is a process of creating a detailed specification for a software module .
- It involves algorithmic design and other implementation specific approaches for a s/w component such as modularity , control hierarchy, data structures etc
- Designers /Technical leads ,senior developers , architects are involved in this phase

Architecture deals with Non functional requirements whereas design deals with functional



Copyright © Capgemini 2015. All Rights Reserved T1

The architecture of a system is its 'skeleton'. It's the highest level of abstraction of a system. What kind of data storage is present, how do modules interact with each other, what recovery systems are in place.

Software design is about designing the individual modules / components. What are the responsibilities, functions, of module X? Of class Y? What can it do, and what not? What design patterns can be used?

So in short, Software architecture is more about the design of the entire system, while software design emphasizes on module / component / class level

6.1.2 Design Phase

Key activities in Design phase

- The design phase includes following activities
 - Identify solution which will meet the customers non functional requirements like performance , security etc..
 - Identify technology stack
 - Identify framework and design pattern
 - Create software architectural overview document
 - Identify major modules and its interfacing with each other as well as external systems if any
 - Defining the logical and physical database model
 - Create test design
 - Plan of the unit and integration test cases
 - Detailing the overall logic of the module in pseudo code or flow charts
 - Detailed database design including constraints data types etc.. (Physical)
 - Detailed interfacing reference (with API and parameters)
 - Prepare design documents



Copyright © Capgemini 2015. All Rights Reserved T2

Architecture constitutes of the following key activities:

- Solution space for “non-functional requirements”
- Decision on Technology Stack
- Framework requirements definition and solution
- Critical decisions for some risky “functional” requirements

Architecture activities are delivered by the Technical Architect and supported by the Design lead **Design** is mainly focused on modeling the functional aspects of an application.

Solution space for “functional requirements” based on defined architecture

Design Pattern choice

Application design

Logical ER Data Model (entities, attributes, relationships)

UML Models - Class, Sequence , Activity etc

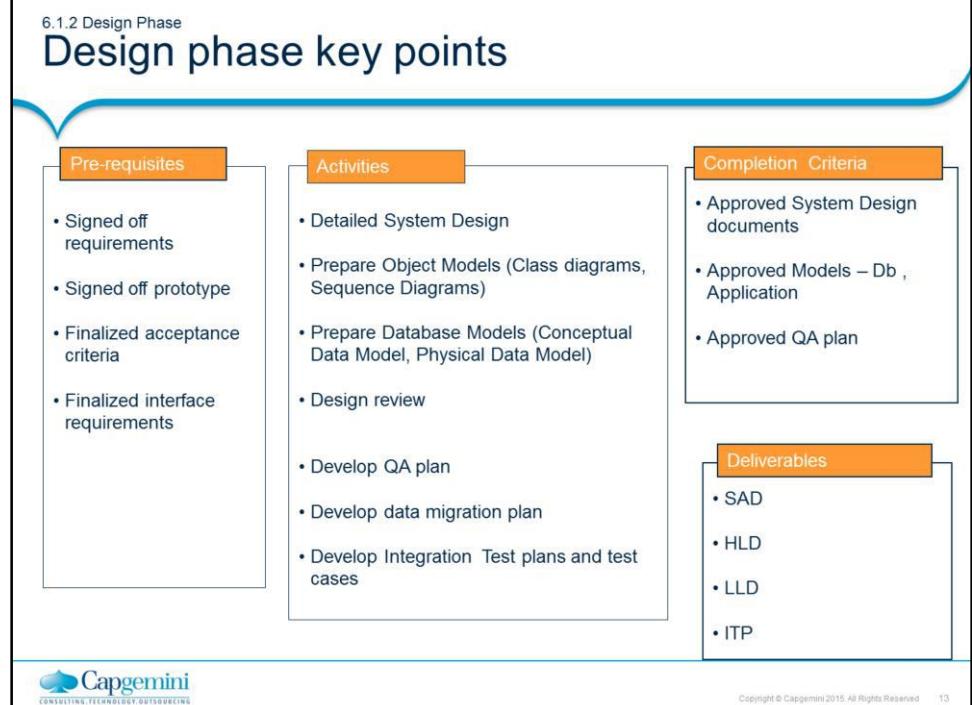
Analysis Model (domain entities, control and boundary classes, their functional attributes and associations)

Additional UML diagrams (as needed)

Data types of attributes

Additional classes, attributes for technical implementation (ex. primary key)

Design activities are delivered by the Design Lead and the Designer .Design Lead is a key role and which acts as a communicator between the architect and designers



6.1.3 Construction Phase

Construction phase

- Also known as implementation phase
- Main objective of this phase is to translate the software design into code , each component identified in design is implemented as a program module following coding guidelines
- Each module in this phase is reviewed and unit tested to determine correct working (White Box testing)
- Unit tested code are then integrated in a planned and a phased manner .
- In each integration step the partially integrated system is tested



Copyright © Capgemini 2015. All Rights Reserved 14

6.1.3 Construction Phase

Construction phase

- In addition to the major activities the following activities are also carried out as well
 - Prepare unit test plan and test case
 - Prepare unit test data
 - Setup coding guidelines
 - Setup the environment for Configuration Management as per CM guidelines
 - Provide suitable environment for base lining code and continuous integration
 - Defect reporting and fixing
- The main role players in this phase are
 - Developers
 - Team Leads



Copyright © Capgemini 2015. All Rights Reserved 15

6.1.3 Construction Phase

Construction phase – key activities

| Pre-requisites | Activities | Completion Criteria |
|---|--|---|
| <ul style="list-style-type: none">Approved design documentsApproved QA planStandard Coding guidelinesReview checklists | <ul style="list-style-type: none">Development environment setupPrepare Unit Test plan and dataBuild CodeCode reviewPerform Unit TestRework and re-testBaseline source code | <ul style="list-style-type: none">Code ready for System testing |
| Deliverables | | <ul style="list-style-type: none">Test reportsBaseline source code |

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 16

6.1.4 Testing Phase

System Testing

- System testing involves testing of all subsystems together
- Also known as Black Box testing It is ideally done by the QA team
- The following types of testing are done as part of system testing
 - Functional testing to validate functional requirements
 - Performance testing to validate non functional requirements



Copyright © Capgemini 2015. All Rights Reserved 17

Goal of functional testing is to test the functionality of the system . The test cases are written from the requirement documents by the QA team as a parallel activity once the requirements are frozen . The system is treated as a black box (implementation independent) .

Goal of the performance testing is to validate the non functional requirement of the system (captured during requirements) , In this kind of testing the system is pushed to its limits to see how it behaves . Some of the performance tests

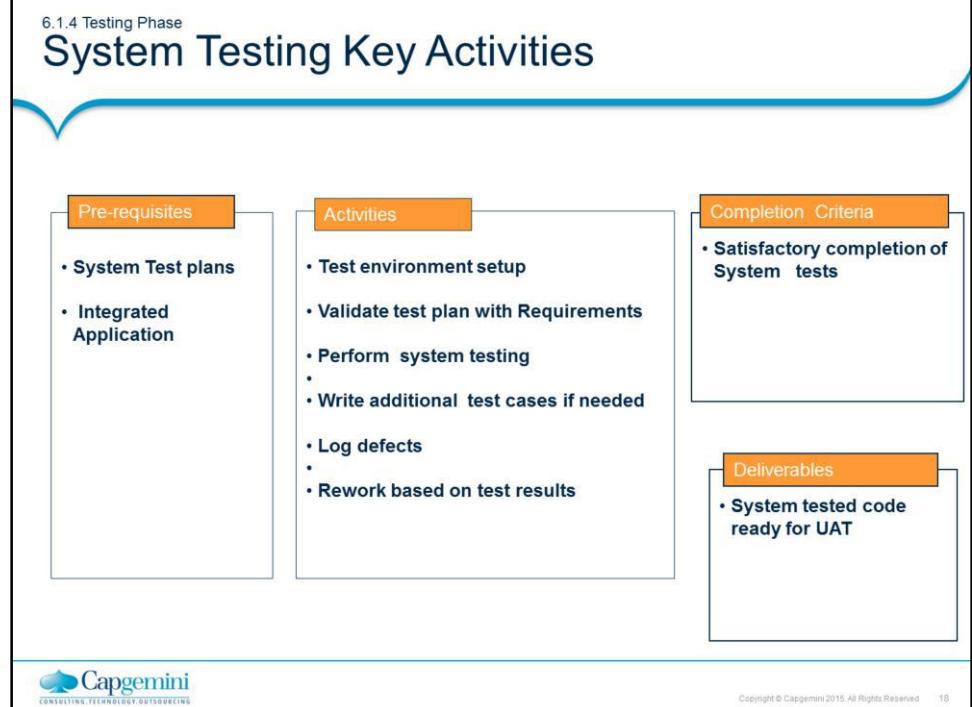
 Stress testing to test stress limits of system (maximum # of users, peak demands etc)

 Volume testing to test large volume of data

 Security Testing to test if the system behavior on security violation

 Recovery Testing to test system's response to loss of data and presence of errors

 Usability testing to test the ease of Use of the system



6.1.5 Testing Phase

Acceptance Testing

- Usually done at the client location by the client , after the findings of System testing is fixed
- Focus of Acceptance test is to evaluate the system's compliance with the business requirements and assess readiness for delivery.
- Acceptance Testing is done in two ways
 - Alpha Testing or Internal Acceptance Testing
 - done by s/w vendors
 - Beta Testing or User Acceptance testing
 - Done by end users of customers or customer's customer
- Outcome of the acceptance testing will enable the user, customers or other authorized entity to determine whether or not to accept the system.



Copyright © Capgemini 2015. All Rights Reserved 19

6.1.5 Testing Phase

Acceptance Testing - Key activities

Pre-requisites

- System and Integration tested code

Activities

- Assist client in setting up testing environment
- Support users / client team, in acceptance testing
- Fix defects / bugs
- Acceptance and signoff from the client
- Assist client team in preparing implementation plan

Completion Criteria

- User accepted application



Copyright © Capgemini 2015. All Rights Reserved 20

6.1.5 Testing Phase

Post Acceptance phase

- After successful acceptance testing plans are made to move the application to the “live environment”
- Activities like knowledge transfer , end user training , project signoff are also done .
- Once when the customers starts using the developed system the maintenance team supports and monitors the system to resolve errors and performance .



Copyright © Capgemini 2015. All Rights Reserved 21

Reviews – What is review ?

- An assessment of a work product created during the software engineering process
- Ensure completeness and consistency of the work product
- Identify needed improvements
- It is a Quality Assurance mechanism to identify discrepancy /deviation from the accepted standards
- Goal of Review
 - To detect and eliminate defects early, effectively and before delivering the product to the customer



Reviews – Why review ?

- Intermediate software products which are not testable as standalone units
- Can find errors not possible through testing
 - E.g., Maintainability: Comments, Consistency, Standards
- Are proactive measure to find out 60-80 % of defects
- Reduce Rework Effort and Improve Schedule adherence
- Enables Quantitative Quality Assessment of any work product



6.2 Review Process

Software Reviews – When , where

- Can happen in all phases of SDLC
- All artifacts can go for reviews
 - Proposals, contracts, statement of work
 - All project work products - Plans, Configuration Mgmt, Test Plans
 - Deliverable and non deliverable work products
 - Software(e.g.: source code) and non software work products (e.g.: documents, test data, etc.)
- Process descriptions
- Policies, brochures, reports, guidelines, standards, training material where required



Copyright © Capgemini 2015. All Rights Reserved 24

6.2 Review Process

Types of Review

- Self Review
 - Done by the author himself with the aid of tools like checklists , review guidelines , rules etc..
- Peer Review
 - Done by “peer” or colleague formally or informally using various approaches
 - Inspection
 - Walk through
 - Pair Programming

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 25

Inspection – It is a more systematic and rigorous type of peer review. Inspections are more effective at finding defects than are informal reviews. In inspection reviewer drives the review process .

Walkthrough – It is an informal review because the work product’s author describes it to some colleagues and asks for suggestions. Walkthroughs are informal because they typically do not follow a defined procedure, do not specify exit criteria, require no management reporting, and generate no metrics.

Pair Programming – In Pair Programming, two developers work together on the same program at a single workstation and continuously reviewing their work.

6.2 Review Process

Review Process

- Input

- Work Product , Specifications, Checklists, Guidelines, Historical Data

- Process

- Prepare for Review
 - Conduct Reviews
 - Analyze Deviations
 - Correct Defects

- Output

- Review Form, reviewed work product,



Copyright © Capgemini 2015. All Rights Reserved 26

6.3 Configuration Process

What is a “Configuration”?

- Arrangement of functional unit of a system in a particular order

Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 27

A configuration is an arrangement of functional units according to their nature, number, and chief characteristics. Often, configuration pertains to the choice of hardware, software, firmware, and documentation. The configuration affects system function and performance.

6.3 Configuration Process

What is Software Configuration Management?

- SCM is the overall management of a software project as it evolves into a software system.
- This includes managing , tracking, organizing, communicating, controlling modifications made in project including release plan
- Also includes the ability to control and manage change in a software project
- Configuration Managers Are responsible for planning the CM activities of their project
- The configuration details of the project are documented in the CMP (Configuration management Plan)



Copyright © Capgemini 2015. All Rights Reserved 28

Why do we need SCM?

- Some of the frustrating problems we face are
 - The latest version of the source code not found
 - A developed and tested feature is mysteriously missing
 - A fully tested program suddenly does not work
 - A wrong version of code was tested
- SCM answers who, what, when and why
 - Who makes the changes?
 - What changes were made to the system?
 - When were the changes made?
 - Why were the changes made?



SCM is the process that defines how to control and manage change. The need for an SCM process is acutely felt when there are many developers and many versions of the software. Suffice to say that in a complex scenario where bug fixing should happen on multiple production systems and enhancements must be continued on the main code base, SCM acts as the backbone which can make this happen.

Without configuration Management the following can happen

- Unorganized project items
- Confused naming conventions
- Review / Delivery of wrong version of code
- Development based on old version of specifications
- No proper access / privilege control; Unauthorized access to secure information
- Redundant file creation
- Change Management becomes ineffective

Elements of SCM

- Configurable item (CI)
 - CI is a collection of items, treated as a unit which are likely to undergo change during the project life cycle and a change to them is likely to affect other CIs.
 - Items that needs to be accessed, controlled, secured and archived is a configurable item
 - (E.g.) Design document, project plan etc..
- Non Configurable item (NCI)
 - Any item / file for which changes need NOT be tracked) i.e. no need to roll back to earlier versions is called a Non-Configured Item.
 - (E.g.) Minutes of Meeting(MOM)



Version:

The term 'version' is used to define a stage in the evolution of a CI, for example versions of source code, etc.

6.3 Configuration Process

Libraries/Folder Structure within configuration management server

- Input Library
- Development Library
- Testing / Review Library
- Release / Delivery Library
- Template Library
- Project Management Library

Tip: The library (folder) can be created on need basis for the project.
No thumb rule to create the same.



Copyright © Capgemini 2015. All Rights Reserved 31

Usage of library - example

▪ Coding and Testing scenario

- Development done in Development library by development team who have access to development folder
- QA team (testing team) would be doing the testing
- As per CM policy QA team wont have permission on Development folder ,
- The code is **moved** from development folder to testing folder
- The code **is moved back** to development folder for rework
- The Re-testing happens in Testing library following the above steps
- Once all the bugs are fixed , the code is moved to release folder .



6.3 Configuration Process

Version Numbering

- A version number is a unique number or set of numbers assigned to a specific release of a software/hardware/firmware
- As updates and new editions of product are released, the version number will increase
- Version numbers are usually divided into sets of numbers, separated by decimal points
- Draft version has X . Y
 - Represents the Version number
 - Changed when there is a
 - CI is completely overhauled
 - Substantial Change
 - Represents the revision number
 - Changed when there is a
 - No change in the overall structure and flow
 - Existing content is minimally



Copyright © Capgemini 2015. All Rights Reserved 33

6.3 Configuration Process

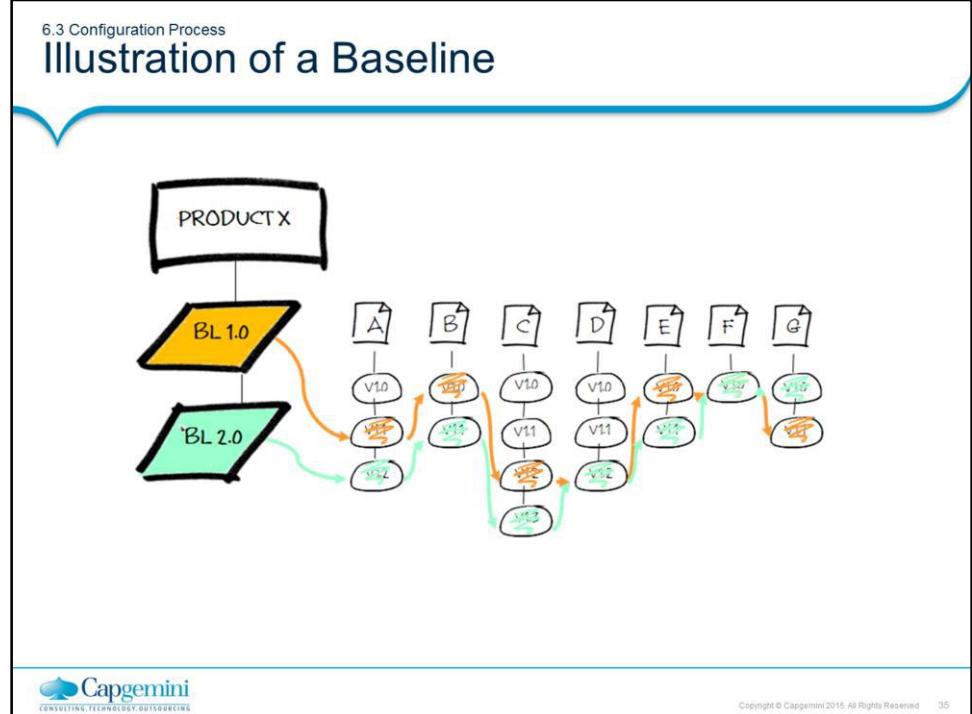
Baselines

▪ Baseline

- A 'baseline' is a CI that has been reviewed and agreed upon and is a basis for further development.
- After base-lining all changes to CI are controlled through a formal change process (Such as Change Management and Configuration Control).
- For example a reviewed and approved Project plan is used as a basis for execution of the project.
- One baseline may have several work product , each having different version number
- Baselinning can be of many types – Input baseline , design baseline , code baseline etc



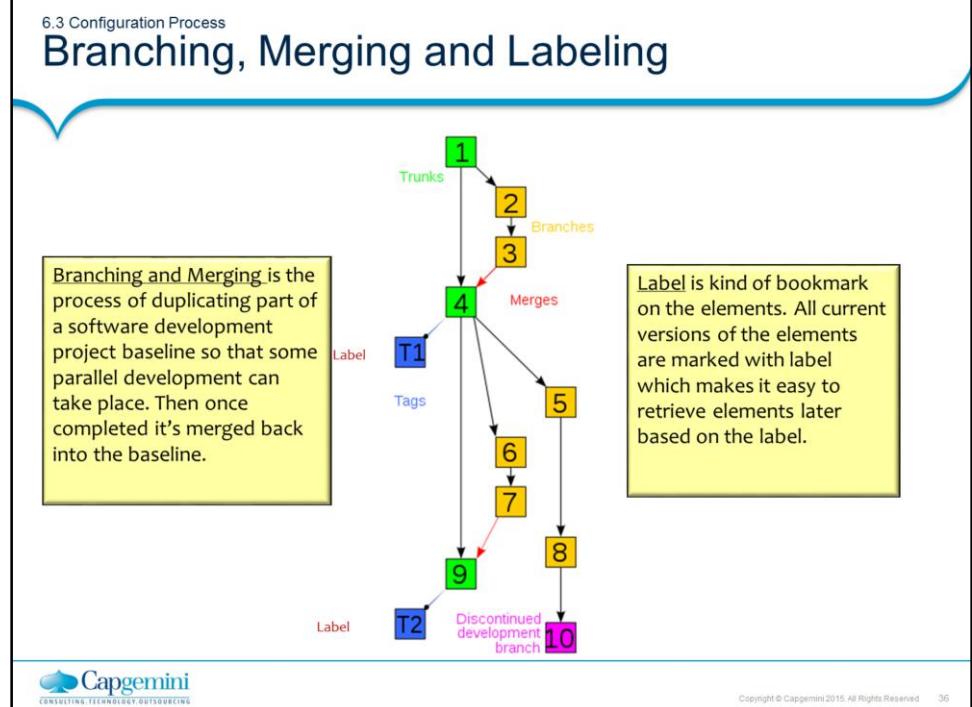
Copyright © Capgemini 2015. All Rights Reserved 34



A baseline defines a set of files, each at a particular version. These need not be the latest (most recent) version. A baseline label uniquely identifies the configuration. Files may belong to one or more baselines.

In the example of Figure 12 baseline BL1.0 is the first baseline recorded. It consists of seven artefacts, each at a unique revision number. For this example, assume that BL1 records the most recent versions of each artefact. As development progresses each artefact is modified as required (that is, some artefact are modified, some are not). At some time later another baseline is taken – BL2.0. In this case BL2.0 records the current latest revisions of each file. Notice that artefact F is unchanged, so F v1.0 is included in both baseline BL1.0 and BL2.0.

In general each successive baseline contains more recent versions of files (but not always).



Branching and Merging are two important aspects of version control. These concepts are extremely useful in parallel software development. The two concepts are briefly explained below

Branch : It is a line of development that exists independently of another line, yet still shares a common history. For example assume we are developing a banking application for American customers. The same application can be used by Canadians with some customization. The solution to this requirement can be achieved by creating a branch for the Canadian customers and incorporating the needed changes. Since the two branches are related, if any changes /bug fixes needed in both can be easily duplicated. The main line of development is called trunk (shown in the diagram), whereas a branch is a side line of a development

Merge : In simple terminologies a merge is basically “copying” the changes across branches. To quote an example , assume that we have started working on the next version of our product (version 4.0) . A critical bug and some minor customization is asked for . To accommodate we create a branch to incorporate change and deploy to the customer. Once the next release is ready we merge the branch completely so as to incorporate the changes done in the branch in the new version

6.3 Configuration Process

Different Roles and Accesses in SCM Tool



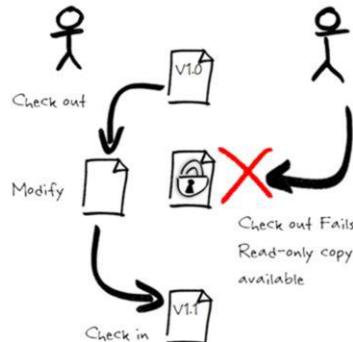
- Administrator
 - All access to all project folders
 - Can add users, create and manage projects and modify their access levels
- Configuration Manager
 - Greater access than all team-members.
 - Creates the basic environment for his projects configuration management
 - Responsible for moving files across projects, establishing baselines, adding requirements files, preparing guidelines, etc
- Team-member
 - Varying access depending on their responsibilities. For e.g. PM gets add/modify access to Management project



Copyright © Capgemini 2015. All Rights Reserved 37

6.3 Configuration Process

Check -in and Check -Out



A check-out is the act of creating a local working copy from the repository. A user may specify a specific revision or obtain the latest.

A check - in is the action of writing or merging the changes made in the working copy back to the repository.



Copyright © Capgemini 2015. All Rights Reserved 38

File locking

In a file locking system only one developer has write access to the artifact. Other developers will have read-only access to the current (stored) version. The file is only available again once it is checked back in.

Summary

- In this lesson, you have learnt about:
 - Different Phases in Software Development
 - Requirements Phase
 - Design Phase
 - Construction Phase
 - Testing Phase
 - Acceptance Phase
 - Review Process
 - Configuration Management Process



Summary



Copyright © Capgemini 2015. All Rights Reserved 39

Review Questions

- Question 1: In _____ phase UNIT Test Plan is written ?
- Question 2: White box testing includes
 - Unit Testing
 - System Testing
 - Operations Acceptance Testing
 - Integration testing
- Question 3: A single baseline may contain many files.(T/F)
- Question 4: A Tester can test in the development library/folder.(T/F)



Copyright © Capgemini 2015. All Rights Reserved 40

Review Questions

- Question 5: SRS (System Requirement Specification) are prepared in _____ phase
- Question 6: _____ and _____ are some non functional requirements
- Question 7: Architecture focuses on functional requirement T/F ?
- Question 8: Alpha testing and beta testing happens in _____ phase ?



Copyright © Capgemini 2015. All Rights Reserved 41