Instructor Notes:

Add instructor notes here.

Instructor Notes:

None

Lesson Objectives

After completing this lesson, you will be able to understand the following topics:

- Importance of exception handling.
- Guidelines for creating exceptions
- Exceptional scenarios in ATM system – case study
- Exception handling case study : Product management system
- What is Defensive programming
- Purpose of defensive programming
- Techniques of defensive programming

Instructor Notes:

Additional notes for instructor

5.1 Importance of exception handling

What is exception handling ?

An exception is an event that occurs during the execution of a program that disrupt its normal course.

- Examples: Hard disk crash; Out of bounds array access; Divide by zero, and so on

No matter how well-designed a program is, there is always a chance that some kind of error will arise during its execution, for example:

- Attempting to divide by 0
- Attempting to read from a file which does not exist
- Referring to non-existing item in array

Exception Handling:

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. Exceptions are used as a way to report occurrence of some exceptional condition. Exception provides a means of communicating information about errors up through a chain of methods until one of them handles it.

- When an exception occurs, the executing method creates an Exception object and hands it to the runtime system — ”throwing an exception”
- The runtime system searches the runtime call stack for a method with an appropriate handler, to catch the exception.

For an example, while booking a ticket in railway reservation system suddenly if the database is down and if an abrupt error page with numerous lines of exception messages displayed on the screen, then user(non-technical person) would be embarrassing and will get annoyed.

Instead of displaying an abrupt error page, if your code catches this database down scenario and displays a graceful message on screen saying "We are currently experiencing some difficulties in our system. Kindly try after some time. Thank you" , then it would be a better message for the end users.

Instructor Notes:

Detailed description is included in the notes page. Kindly ask participants to look into the same.

5.1 Importance of Exception Handling

Importance of Exception Handling

Exceptions may occur at runtime and disrupt the application flow.
Unexpected events happen and the programmer should always be prepared for the worst.
Use exception handling for separating Error-Handling Code from "Regular" Code
Providing meaningful error messages

Importance of Exception Handling

- Once an exception occurs at runtime, the program execution will be stopped immediately and an exception message gets displayed which may not be understandable by all the end users.
- Whenever an unexpected input is passed to an application by the end users, program may get crashed. So in order to create an application which should be sustainable at any scenarios, exception handling is mandatory to be taken care.
- Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program. Enclose the error handling code separately in try block from regular code for making program execution to happen without any disruption even though an exception occurs.

Instructor Notes:

An example included in the notes page.

5.2 Guidelines for creating exceptions

Guidelines for creating exception handlers

We need to think of all possible conditions from the case study that are likely, unlikely, and impossible.
We need to analyze the consequences of failures along with the probability.
Accordingly create an exception for each of the possibility
Document the reason for the exception
Raise the exception whenever the condition arises in the sub module
Catch the exception in the appropriate parent module

- Pseudocode example without taking care of exception's

```
SUB readFile
        open the file;
        determine its size;
        read data from the file;
        close the file;
END SUB
```

At first glance, this function seems simple enough, but it ignores all the following potential errors.

- What happens if the file can't be opened?
- What happens if the length of the file can't be determined?
- What happens if the read fails?
- What happens if the file can't be closed?

Instructor Notes:

An example included in the notes page.

See the below Pseudocode with exception handling for taking care of the possible potential errors.

```
SUB readFile
        open the file;
        determine its size;
        read data from the file;
        close the file;
EXCEPTION
        WHEN fileOpenFailed THEN
                doSomething;
        WHEN sizeDeterminationFailed THEN
                doSomething
        WHEN readFailed THEN
                doSomething;
        WHEN fileCloseFailed THEN
                doSomething
END SUB
```

Instructor Notes:

None

5.3 Exceptional scenarios in ATM system – case study

Case Study 1

Case 1: Consider the transaction at a Bank ATM. An user needs to withdraw some money.

- Analyze this transaction from the user's point of view.
- Think out all scenarios about what can go wrong.
- Determine how the system should perform this transaction.
- What irritations have you faced at an ATM?

Scenarios:

- User authentication: User thinks he has entered the correct password. However, the system says "invalid password".
- The ATM does not provide coins. However, it expects the user to enter the amount in the format 5000.00
- User enters the amount as 15000, and the system gives 150 notes of hundred rupees
- The system asks the user for a password, then asks the details about the transaction (like amount, requirement of a printed receipt, etc.), and then at the end it tells the user that the password is invalid.

Instructor Notes:

None

5.3 Exceptional scenarios in ATM system – case study

Case Study 1

Case 2: Now, consider the same transaction from the programmer's point of view.

- Visualize what all can go wrong.
- Determine how should the system respond.

Scenarios:

- The user may have multiple ATM cards. Although the correct password has been entered, it may be for a different card. Does the error message consider this possibility?
- After one transaction is completed, the system says "Thank You", without checking whether another transaction is desired.
- Does the system provide multiple alternatives or options to recover from errors, or to deal with various contingencies?
- Notes of a specific denomination are not available. Can user enter "OK" to ask for another denomination?

Instructor Notes:

None

5.3 Exceptional scenarios in ATM system – case study

Case Study 1

Machine is out of cash. What options are available?

- Allow for withdrawal of a smaller amount.
  - The system should suggest this amount, and not let the user guess by trial and error!
- Display the location of another ATM nearby.
  - Can we double check whether that ATM has enough cash?

What if the communication link is very slow (during "Authentication" and "Balance Check")?

What if we get a device error, file io error, or disk full error while writing the transaction to the database?

Instructor Notes:

None

5.4 Exceptional scenarios

Common Issues

Some of the common issues in Exception Handling are:

- On opening a file for read access, you get an exception "file does not exist".
  - Check for this condition.
  - Display a specific error message, with the name of the file that was not found.

Some of the common issues in Exception Handling are (contd.):

- On opening a file for write access, you get an exception "file already exists".
  - Check for this condition.
  - Display a specific error message, with the name of the file that already exists.
- In either case, analyze if it is necessary to stop the program! Or is it possible to allow the user to provide the correct file, and continue execution?

Instructor Notes:

None

5.4 Exceptional scenarios

Common Issues

Some of the common issues in Exception Handling are (contd.):

- On any DBMS operation, always check for successful completion, or error returned.
  - Complex software can fail in many different ways.
    - For example: access problem, concurrency problem, integrity problem, etc.
- For code involving "memory allocation", always check for failure to allocate memory.
  - This is always possible if the application has been running for many hours.

- Some of the common issues in Exception Handling are (contd.):
  - In computations, ensure that there is no "divide by zero".
  - While using strings, ensure that they are NULL terminated.
  - In languages that support exception handling, ensure you understand how it works. Then use the appropriate constructs like Assert, Throw, Catch, etc.

Instructor Notes:

None

5.4 Exceptional scenarios

Common Issues

Some of the common issues in Exception Handling are (contd.):

- Check for exceptions at the interfaces between two modules.
  - Is the "calling module" making any assumptions that the "called module" does not guarantee?
  - Is the "called module" making any assumptions that the "calling module" may violate?
- For example: Suppose that a module performs a "binary search" on an array, which is given as input to the module. Does it assume that the array is sorted? Does the interface definition for this module clearly document about that?

Instructor Notes:

None

5.4 Exceptional scenarios

Common Issues

Some of the common issues in Exception Handling are (contd.):

- In object oriented languages, check for exceptions with respect to the collaborations between objects.
- Exceptions can either be "handled by a module" or "reported to a higher level module as a return value or parameter".
  - This should be decided based on the level at which it is possible to take appropriate recovery actions.

For example:

- In a Railway Reservation system, if the desired seating is not available, check:
  - whether alternate seating options will be acceptable
  - whether alternate date is acceptable
  - whether alternate train or route will be acceptable

Instructor Notes:

None

5.5 Exception handling case study : Product management system

Case Study 2

Suppose you are creating the applyDiscount module which applies discount to the price of an existing product available in the products database

Pseudocode of "applyDiscount" and "getProductPrice" Module :

```
SUB applyDiscount(productId,discount)
        PRINT getproductPrice(productId) * discount;
END SUB

SUB getProductPrice(productId)
        RETURN productPrice;
END SUB
```

applyDiscount module is used to apply discount on a productprice for a particular product.

The applyDiscount module includes a call to the getproductPrice module which searches for the required product based on the productId and return the price of a product

Instructor Notes:

None

5.5 Exception handling case study : Product management system

Case Study 2

Should the productPrice be returned from the getProductPrice module if an invalid productId is entered?

Revised Pseudocode of "getProductPrice " module for handling exception if product does not exist.

```
SUB getProductPrice(productId)
        IF elementfound(productId) THEN
            RETURN productPrice
        ELSE
            RAISE NoSuchElement("Product doesn't exist with the id"+ productId)
        END IF
END SUB
```

Consider "NoSuchElement" is an user defined exception used here for throwing exception when a product doesn't exist with a given productId.

Description of the pseudocode given in slide

- If the product id supplied to the productPrice module does not exist in the database then an exception, "Product does not exist along with productId", should be raised from the getproductPrice module
- The applyDiscount module should catch and handle the exception appropriately
- Rasie is a keyword which can be used to throw an exception.
    - Syntax: RAISE exceptionname(message);
    - Once if an exception raised from a module, then catch the exception and display the exception message while invoking the module from which the exception is thrown.
    - Exception messages should be more meaningful along with the entered productId.

Instructor Notes:

None

5.5 Exception handling case study : Product management system

Case Study 2

Revised Code of applyDiscount and getProductPrice Module with exception handling

```
SUB applyDiscount(productId,discount)
        PRINT getProductPrice(productId)*discount
EXCEPTION
        WHEN NoSuchElement THEN
                PRINT errormessage //Errormessage returned from exception
END SUB
SUB getProductPrice(productId)
        IF elementfound(productId) THEN
            RETURN productPrice
        ELSE
            RAISE NoSuchElement("Product doesn't exist with the id"+ productId)
        END IF
END SUB
```

EXCEPTION section will be executed whenever exception occurs. Using WHEN section exception can be handled.

If the given productId exists in database, then updated price will be returned else "Product doesn't exist with the given productId " message will be printed.

Instructor Notes:

None

5.5 Exception handling

Exception Handling

Depending on the “degree of criticality”, the developer is required to design the exception handlers:

- to give meaningful error messages, or
- to provide alternatives and options, or
- to provide recovery options

Instructor Notes:

Additional notes for instructor

5.6 Defensive Programming

What is Defensive Programming

Defensive programming – program tries its best to provide service despite errors or unexpected conditions

- It is like defensive driving. "Should you drive on the assumption that all other drivers will follow the rules?" or, "Should you play safe by assuming they will do the unexpected at least once in a while?"
- Similarly, good programmers do not assume that users will always do the expected thing, i.e. pass the right type of parameters, initialize variables, etc.

Defensive programming enables us to detect minor problems early on, rather than get bitten by them later when they've escalated into major disasters.

Defensive programming is a method of prevention, rather than a form of cure. Compare this to debugging—the act of removing bugs after they've bitten. Debugging is all about finding a cure.

Defensive Programming is not

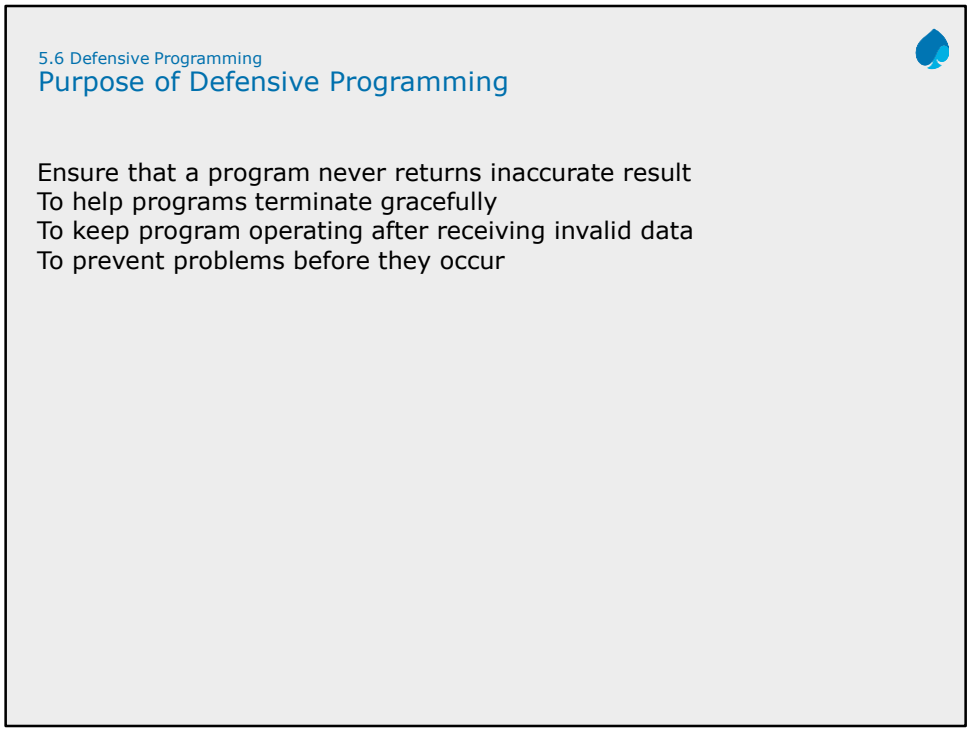
Error checking

Testing

Debugging

Instructor Notes:

Additional notes for instructor



Purpose of Defensive Programming

Ensure that a program never returns inaccurate result even though valid data is passed.

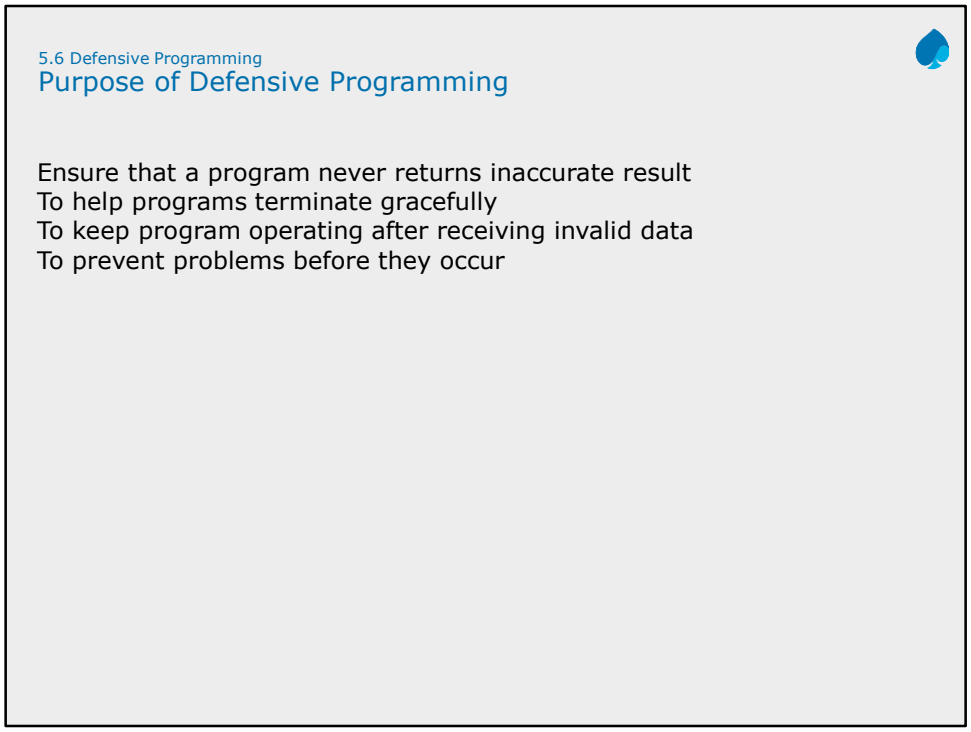
Abnormal termination of the program will be avoided

Even though, invalid data is passed instead of abnormal termination of program execution, meaningful error messages has to be displayed.

Thinking all the possible and impossible scenarios and take care of exception to prevent problem occurrence.

Instructor Notes:

Additional notes for instructor



Purpose of Defensive Programming

Ensure that a program never returns inaccurate result even though valid data is passed.

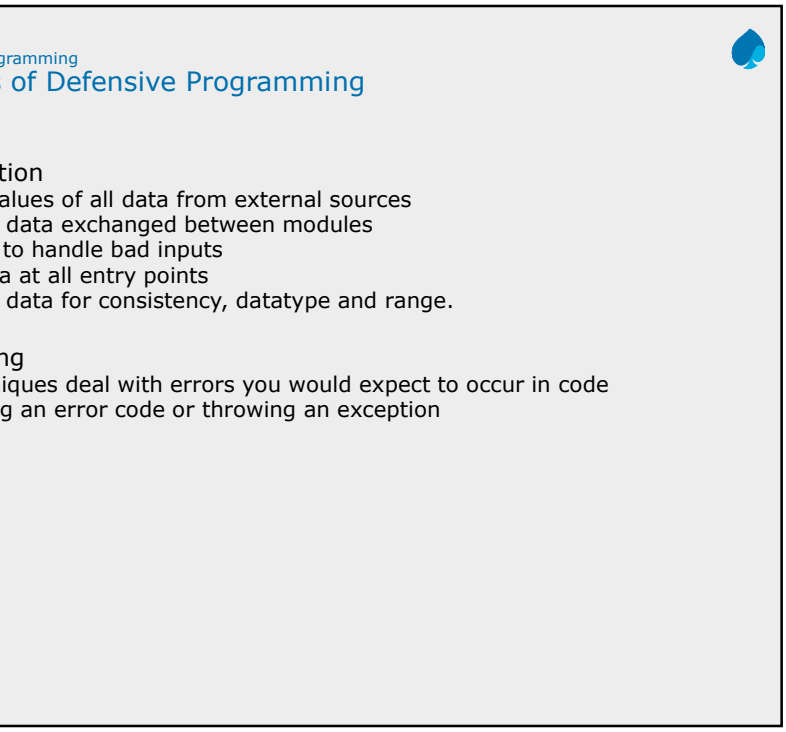
Abnormal termination of the program will be avoided

Even though, invalid data is passed instead of abnormal termination of program execution, meaningful error messages has to be displayed.

Thinking all the possible and impossible scenarios and take care of exception to prevent problem occurrence.

Instructor Notes:

None



Techniques of Defensive Programming

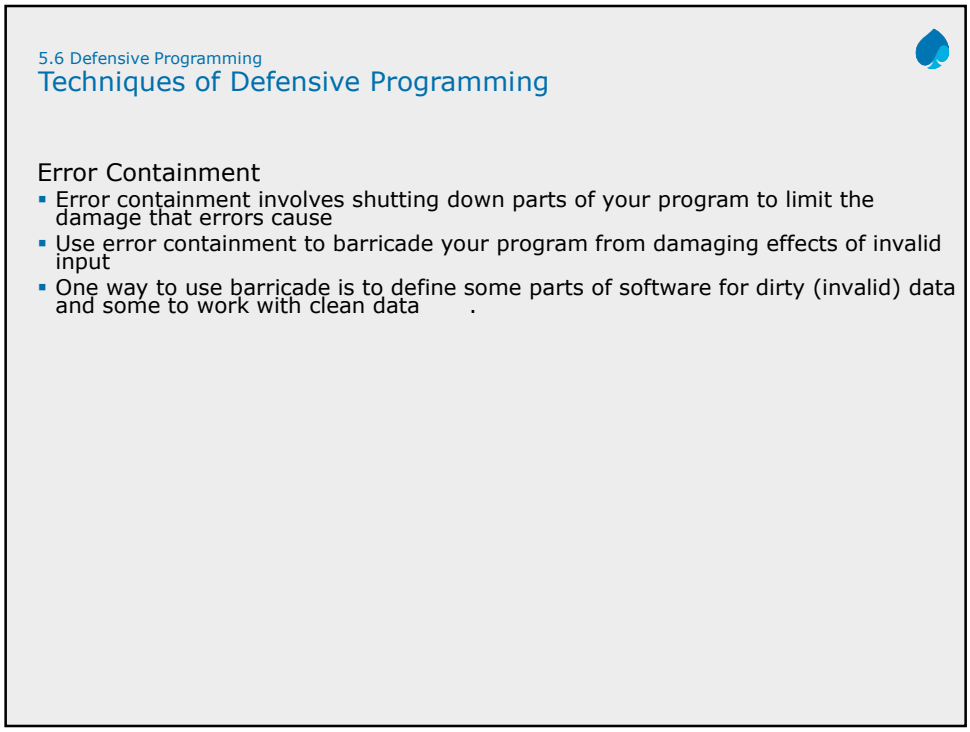
1. **Input validation**

- **Check the values of all data from external sources :** When getting data either from a file or from a user, check to be sure that the data falls within the allowable range and correct type of value is accepted.
- **Validate the data exchanged between modules:** Checking the values of routine input parameters is essentially the same as checking data that comes from an external source
- **Decide how to handle bad inputs:** Once you have detected an invalid input, take care of necessary technique to display an error message or return error code.
- Validate the data for
    - **consistency :** The consistency check compares new data with previous data. For example, a current meter reading against past meter readings.
    - **data type :** The data type check ensures input data is of the correct data type. For example, numeric or alphabetic.
    - **range :** check ensures that input data is within a specified range.

2. **Error handling :** these techniques deal with errors you would expect to occur in code

Instructor Notes:

None



Techniques of Defensive Programming(Contd..)

Example for Error Handling:

```
IF (fileExists)  THEN
        Read the data from the file
ELSE
        errorCode = -1; //File doesn't exists
        return errorCode
END IF
```

3. Error Containment

- Protect your code from an invalid data coming from "outside"(Data from an external system or the user or a file)
- Establish "barricades"(module where validation logic exists) for leaving dangerous data outside of the boundary(like before invoking module), everything inside of the boundary(within module) is safe. For an example, send input parameters to a module only if it is valid(Inside of the boundary is safe)
- In the barricade code(isValid Module), validate all input data (check all input parameters) for the correct type, length, and range of values. Double check for limits and bounds.
- **For an Example:**

```
IF(data is invalid) THEN
        PRINT error message
ELSE
        Invoke a module to execute logic
END IF
```

Instructor Notes:

5.6 Defensive Programming

Demo : Exception Handling and Defensive Programming

Refer the pseudo code for calculating price after applying discount in which exceptions handlers are used.

- ExceptionHandling

Refer the pseudo code for calculating price after applying discount in which defensive programming techniques are applied.

- DefensiveProgramming

Pseudocode to calculate price after applying discount .

The below defensive programming techniques are applied.

Input Validation

ProductId and discount value should accept only digits

ProductId should already exists

Error Handling

Error code(-1) will be returned if productId doesn't exists

Error Containment

applyDiscount module will be invoked only if the given inputs are valid else error message will be printed

Instructor Notes:

Lab

Exception Handling Lab Exercises - Lab 4

Write pseudocode to test the concept of handling exceptional scenarios.

Instructor Notes:

Summary

Exceptions are powerful error handling mechanism

Developer is required to design the exception handlers:

- to give meaningful error messages, or
- to provide alternatives and options, or
- to provide recovery options

Defensive programming is used to ensure the continuing function of a piece of software under any circumstances.

Techniques of Defensive Programming

- Input validation
- Error Containment
- Error handling

Instructor Notes:

Question 1 : A,B,C
Question 2. B,C,D

Review Questions

Question 1 : What is the main purpose of input validation ?

- A. To ensure that data exists in a field
- B. To ensure that input data is of correct datatype
- C. To ensure that input data is within a specified range
- D. To ensure that pre input check data is correct

Question 2 : What is the purpose of defensive programming ?

- A. Ensure that a program never returns inaccurate result
- B. To help programs terminate gracefully
- C. To keep program operating after receiving invalid data
- D. To prevent problems before they occur

Instructor Notes:

Question 3. D
Question 4. A

Review Questions

Question 3 : What is the purpose of exception handling ?

- A. To provide meaningful error messages
- B. To provide alternatives
- C. To separate error code from regular code
- D. All of the above

Question 4 : An exception is an event that occurs during the execution of a program that disrupt its normal course.

- A. True
- B. False