Error Handling.

- Symbx error - error in code itself
- Run time error - like writing an a Speed Stock that did not exist program suddenly stops
- A logic error - human error → wrong input / selection

Debugger

- Executes program code one line at a time and allows us to examine the content of one or more variables/properties they change
- Breakpoints → insert a breakpoint in code where you want the program to stop and lets you examine variables/properties
- Debugger allows user to execute which line they want, jump in out of loops etc

Watch Window

- The locals window is useful for examining the state of certain variables and properties as they are evaluated by the compiler, but items in the locals window are only shown for the current statement (the highlighted statement)
- To view contents of variables and properties throughout the execution of a ~~statement~~ program, we use the watch window.

Error handler:

- Use error handler in any situation where an action has the potential to produce an error (and stop) program execution

2

The heart of error handling is the "On Error" statement. This statement instructs VBA what to do when an a run time error is encountered, Takes three forms:

On error goto 0
On Error Resume Next
On Error Goto <label>:

On Error Goto 0 - default mode in VBA. When a run time error occurs VBA should display its standard run time error message box, allowing you to enter debug mode

On error Resume Next : most commonly used and misused form instructs VBA essentially to ignore the error and resume execution of next line of code IT DOES NOT FIX an error.

On error Goto <label> tells VBA to transfer execution to the line following the specified line label. When an error occurs, code execution immediately goes to the line following the line label

```
On error Goto ErrHandler:
  code
exit sub
ErrHandler:
  code
  Resume Next
End sub
```

Resume used to exit error handler. Can be
Resume Resume Next Resume <label>
Resume on its own jumps back to statement that caused error.

Resume next jumps back to the after error line and executes from there

3

<> (not equal)

Resume <label> jumps back to code block with the label

Debug.print - allows you to write output to the immediate window. Put into certain parts of code where you would like value of variable without stopping execution

Debug.Assert - Stops your code when the boolean value is passed to it evaluates to false Debug.Assert x <> 5, stops when x = 5.

Stop Statement - Alternate to Debug.Assert using a Stop Statement inside an IF clause:

```
If x = 5 Then Stop
```

Other Tips:

- Use option explicit at beginning of module
- Format code with indention
- Use comments
- Keep Sub and Function Seperate
- Use macro recorder to help identify properties and methods
- Use debugger

Error Handling Java - Eclipse

Exceptions are irregular or unusual events that happen in a method the program is calling which usually occurs at runtime

Try and catch Statement

```
try {

} catch (exception) {
}
```

The try and catch method runs the code inside the try statement. Once an error is found it jumps into the catch statement.
- The catch statement deals with the error.

Throw.

- We can use a throw exception inside the code when we manage the "throw" an exception with a specific name
- The catch statement catches this statement and deals with the thrown error.
- Can catch multiple errors by extending code

Catch block is an exception handler and handles the type of exception indicated by its argument.

- Otherwise java will raise a default error which is usually

Debugger In Eclipse for Java

Allows you to run a program interactively while watching the source code and the variables during execution.

Breakpoint in the source code specify where execution should stop

To stop execution only if a field is read or modified you can specify watchpoint