UI Software Organization
IAT351

Week 2 Lecture 1
16.01.2008

Lyn Bartram
lyn@sfu.ca

Some slides adapted from K. Edwards, Georgia tech

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Today's agenda

- What is the UI? A whirlwind tour of history
- Software organisation and UI architectures
  - Toolkits!
- Introduction to the Swing toolkit and some examples

Foundations of HCI

- Understanding where you've come from can help a lot in figuring out where you're going
- Knowledge of an area implies an appreciation of its history

Paradigms

- Predominant theoretical frameworks or scientific world views
  - e.g., Aristotelian, Newtonian, Einsteinian (relativistic) paradigms in physics
- Understanding HCI history is largely about understanding a series of paradigm shifts
  - Not all coming on next slides are really "paradigm" shifts, but you get the idea
  - Critical technology design depends on vision
  - Incremental development often reduced to habit, availability or marketing ..

Paradigm Shifts

- Cards,tape -> VDU
- Mainframe -> PC
- Glass tty -> WIMP interface
- Commands -> Direct manipulation
- Direct manipulation -> Agents
- Visual -> Multimedia
- Linear -> Web-like
- Desktop -> Ubiquitous, Mobile
- Single user -> CSCW
- Purposeful use -> Situated use

History of HCI

- Digital computer grounded in ideas from 1700’s & 1800’s
- Technology became available in the 1940’s and 1950’s

Evolution of computing

- 1950s - 1960s
  - Computers appeared on the commercial scene
  - Difficult to use, cumbersome
  - Ran in “batch-mode”, experienced operators
  - Cards
- Early 1960s - 1980s
  - Timesharing systems
  - Manual command line

Evolution of computing

- 1970s
  - First personal computers
  - Raster graphics-based networked workstations
  - Mouse pointing devices, desktop metaphor, windows, icons
  - WIMP
  - Widespread adoption
  - Man-machine interface (MM!)
- mid 1980s - now
  - Human-Computer interaction (HCI)

Vannevar Bush

- "As We May Think" - 1945 *Atlantic Monthly*

  "…publication has been extended far beyond our present ability to make real use of the record."

Vannevar Bush

- Postulated **Memex** device
  - Can store all records/articles/communications
  - Large memory
  - Items retrieved by indexing, keywords, cross references
  - Can make a **trail** of **links** through material, etc.
- Envisioned as microfilm, not computer

J.R. Licklider

- 1960 - Postulated “man-computer symbiosis”

- Couple human brains and computing machines tightly to revolutionize information handling

Vision/Goals

- Immed
  - Time sharing
  - Electronic I/O
  - Interactive, real-time system
  - Large scale information storage and retrieval

Intermed

- Combined speech recognition, character recognition, light-pen editing

Long-term

- Natural language understanding
- Speech recognition of arbitrary users
- Heuristic programming

Mid 1960's

- Computers too expensive for individuals -> timesharing
  - increased accessibility
  - interactive systems, not jobs
  - text processing, editing
  - email, shared file system
  - Single, dedicated task

Need for HCI

Ivan Sutherland

- **SketchPad** - '63 PhD thesis at MIT
  - Hierarchy - pictures & subpictures
  - Master picture with instances (ie, OOP)
  - Constraints
  - Icons
  - Copying
  - Light pen as input device
  - Recursive operations

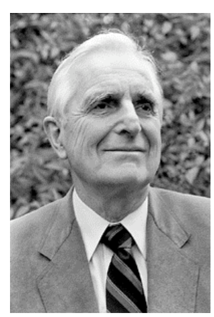
Video Display Units

New paradigm

- More suitable medium than paper
- Sutherland’s Sketchpad as landmark system
- Computers used for visualizing and manipulating data

Douglas Engelbart

- Landmark system/demo:
  - hierarchical hypertext, multimedia, mouse, high-res display, windows, shared files, electronic messaging, CSCW, teleconferencing, ...

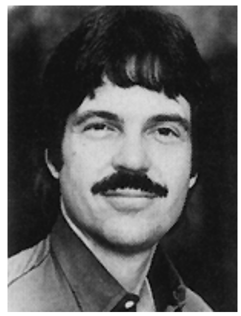


Inventor
of the mouse

Alan Kay

- Dynabook - Notebook sized computer loaded with multimedia and can store everything

Personal computing



Desktop interface

Personal Computing

New paradigm

- System is more powerful if it's easier to use
- Small, powerful machines dedicated to individual
- Importance of networks and time-sharing
- Kay's Dynabook, IBM PC

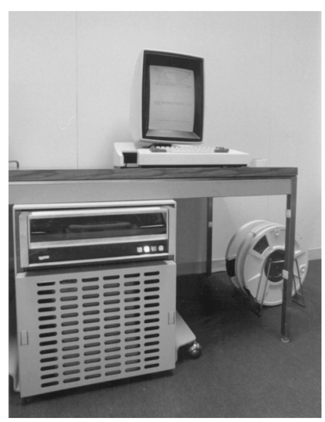
Personal Computers

- ‘70’s IBM PC and the command line UI
  - Text and command-based: symbolic input
  - Monochrome
  - Recall not recognition
  - Required new control mappings and modes
    - Different across applications
  - Single input modality, serialised effort
  - Hard to learn - but efficient for experts
  - Small spatial/discrete capability (remember Rogue?)
  - Modal input

PCs with GUIs

Xerox PARC - mid 1970’s

- **Alto**
  - local processor, bitmap display, mouse
  - Precursor to modern GUI, windows, menus, scrollbars
  - LAN - ethernet

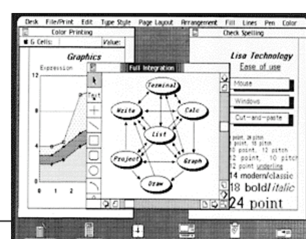


Xerox Star - '81

- First commercial PC designed for "business professionals"
  - desktop metaphor, pointing, WYSIWYG, high degree of consistency and simplicity
- First system based on usability engineering
  - Paper prototyping and analysis
  - Usability testing and iterative refinement
- Advent of the 2D device in variable use
  - Pointing, selection, control

Apple Lisa - '82

- Based on ideas of Star
- More personal rather than office tool
  - Still $$$
- Failure



Apple Macintosh - '84

- Aggressive pricing - $2500
- Not trailblazer, smart copier
- Good interface guidelines
- 3rd party applications
- High quality graphics and laser printer

WIMP

- **W**indows, **I**cons, **M**enus, **P**ointers
- Can do several things simultaneously
  - Context switching
  - Start of the religious wars on tiled vs overlapping
- Familiar GUI interface - desktop metaphor
- Xerox Alto, Star; early Apples
- Used a mouse and a keyboard for input

Ben Shneiderman

- Coins and explores notion of direct manipulation of interface
- Long-time Director of HCI Lab at Maryland

Direct Manipulation

New paradigm

- ‘82 Shneiderman describes appeal of graphically-based interaction
  - object visibility
  - incremental action and rapid feedback
  - reversibility encourages exploration
  - replace language with action
  - syntactic correctness of all actions
- WYSIWYG, Apple Mac

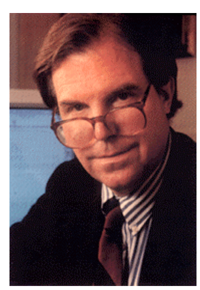
Multimodality

New paradigm

- Mode is a human communication channel
  - Not just the senses, e.g., speech and non-speech audio are two modes
- Emphasis on simultaneous use of multiple channels for I/O
- Fragmentation and integration across many interaction channels
  - Multimodal != additive ??
- More intuitive ?

Nicholas Negroponte

- MIT machine architecture & AI group ‘69-’80s
- Ideas:
  - wall-sized displays, video disks, AI in interfaces (agents), speech recognition, multimedia with hypertext



Language (Agents)

New paradigm

- Actions do not always speak louder than words
- Interface as mediator or agent
- Language paradigm
- Different communication mapping

CSCW

New paradigm

- Computer-Supported Cooperative Work
- No longer single user/single system
- Micro-social aspects are crucial
- E-mail as prominent success but other groupware still not widely used
  - Move to real-time and both f2f and remote
- Singular and shared interaction environments
  - Stanford iRoom
  - Multiple mice - I machine (Inkpen)
  - Remote interaction techniques
- WYSIWIS

Mark Weiser

- Introduced notion of “calm technology”
  - It’s everywhere, but recedes quietly into background
  - Ubiquitous computing
- CTO of Xerox PARC
- Sensors, actuators
- Vision and image processing

Ubiquity

New paradigm

- Person is no longer user of virtual device but occupant of virtual, computationally-rich environment
- Can no longer neglect macro-social aspects
- Late ‘90s - PDAs, VEs, ...
- 2000 - cell phones, RFID, tangible Uis …
- Large and small shared and partitioned devices
- Information is no longer device-singular in an application
  - Uniformity of techniques no longer applies?
  - Your machine - our information?

Immersive and manyD environments

- Immersive Virtual reality (NSCA Cave™)
  - Fred Brooks
  - Henry Fuchs
  - Very specialised and hard to use
- Fishtank VR and augmented reality
- Single or combined devices with many DOF (head tracker, Flock of Birds)
  - Human factors of many DOF are challenging
- Stereo and large displays
  - Increasingly common usage especially in CSCW

Our bodies, our interaction devices …

- Instrumenting the human
  - Eye tracking/head tracking
  - Motion capture
  - High resolution direct input
  - Less cognitive load?
- Haptic and physical interfaces
  - Using touch and force for direct input
  - Sensors and other capture for indirect input (biomechanical signals-GSR)
  - Tangible bits - Ishii
  - The interactive floor
- There's lots more coming….

So how do we construct these user interfaces?

Architectural goals

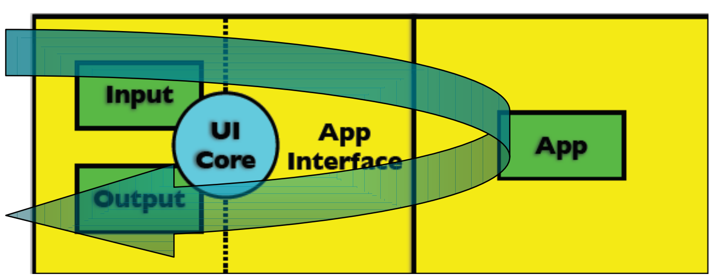
1. Separation of concerns
   - We generally want to think of the "UI" as only one component of the system
2. Multiplicity of presentation options
   - Pluggable, quasi independent views
3. Coordination for interaction
   - Coherent framework for mapping and controlling input to logic to output

But in practice

- Separation of concerns is a central theme of UI organisation
  - Continual challenge
  - Tradeoff between goals (2) - (3) and (1)
  - Real separation is almost a lost cause
- Nature of interaction dictates architecture
  - Highly interactive, responsive, multimodal system requires specialised application interface

Conceptual overview of the UI

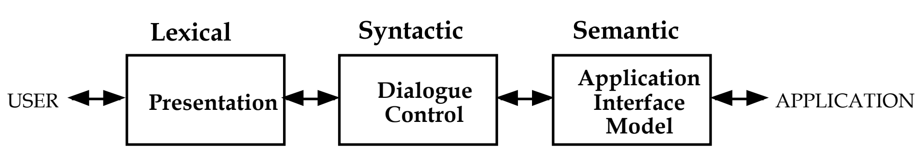
Basic UI Flow



How would you architect this?

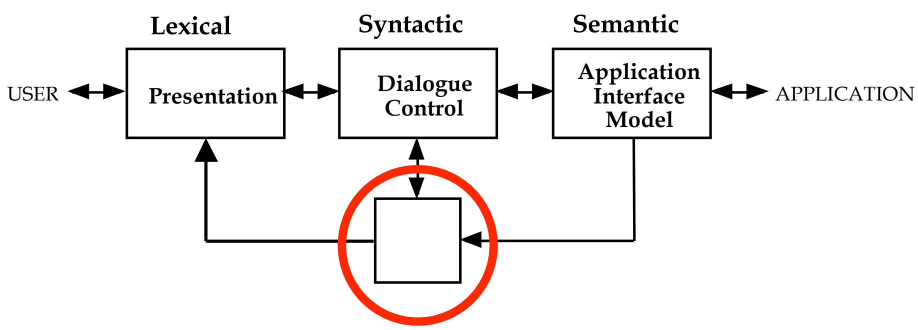
- Tempting to architect the system around these boxes
  - One module for input, one for output, etc
  - Has been tried (the “Seeheim model”)
  - Didn’t work well

Seeheim model



Result of 1985 workshop in Seeheim, Germany
Basis of the UIMS approach

Problem:
Rapid Semantic Feedback



Model was too linear

- General Flow
  - Prompt user for input
  - Program reads in a line of text
  - Program runs for a while (user waits)
  - Maybe some output
  - Loop back to beginning
- Not very interactive
  - only gives output after user does something
    - with long wait cycles
  - does not work well for graphical and interactive apps
  - impossible to create a word processor here
    - want to allow printing, inserting, whenever user wants

Big box architectures don't work well because...

- Modern interfaces: set of quasi-independent agents
  - Each "object of interest" is separate
  - e.g. a button
    - produces "button-like" output
    - acts on input in a "button-like" way
    - etc.
  - Each object does its tasks based on
    - What it is
    - What its current "state" is
      - Context from prior interaction or application

The philosophical shift

- Compiler mentality
  - Lexical/Syntactic/Semantic
  - Seeheim, ARCH
- Object mentality
  - Interface as collection of objects

Object-based architectures

- *Interactor* objects (“object of interest”)
  - AKA components, controls, widgets
  - Example: an on-screen button
- Each object implements each aspect
  - Common methods for
    - Drawing output (button-like appearance)
    - Handling input (what happens when button is clicked)
- Objects organized hierarchically
  - reflecting spatial containment relationships
  - Reflecting behaviour flow

Modern implementation support

programming tools

- levels of services for programmers

1. windowing systems
2. Dialogue control
3. interaction toolkits
4. user interface management systems (UMIS)

Implementation support

1. windowing systems
   - Device independence
   - Multiple tasks (simultaneous, distinct user activity)
2. Dialogue control
3. interaction toolkits
4. user interface management systems (UIMS)

Implementation support

1. windowing systems
2. Dialogue control
   - Modal, tight “read-evaluate-act” loop
   - Notification or event-based
   - Paradigm for how application is controlled
3. interaction toolkits
4. user interface management systems (UIMS)

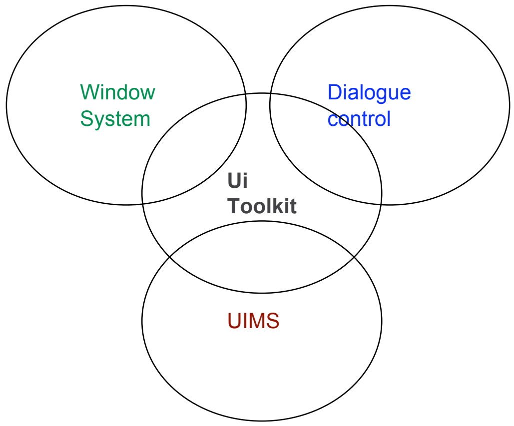
Implementation support

1. windowing systems
2. Dialogue control
3. interaction toolkits
   - Programming interaction objects and behaviours (UI toolkits)
   - Component-based systems
   - UI “builders”
4. user interface management systems (UIMS)

Implementation support

1. windowing systems
2. Dialogue control
3. interaction toolkits
4. user interface management systems (UIMS)
   - Conceptual architectures for separation
   - Techniques for expressing dialogue

Implementation support



UI Toolkits (GUI Toolkits)

- System to provide development-time and runtime support for UIs
  - Event- driven programming
  - Widgets/components
  - Interactor tree
- Specific interaction techniques
  - Libraries of interactors
  - Look and feel
- How the UI connects to the application (the API)
- Describes how most GUIs work
- We will be using SWING, the Java GUI toolkit
- We will not be using UI builders

Toolkit detail (roadmap)

- Core functionality
  - Hierarchy management
    - Create, maintain and tear down the tree/graph of interactor objects
  - Geometry management
    - Dealing with coordinate systems
    - Windows and graphics
  - Interactor status
- Output/display
  - Layout
  - Drawing and redrawing (damage management)
  - Images and text

Toolkit detail (roadmap)

- Input
  - Picking
    - Figuring out what interactors are active under a given screen point
  - Events
    - Dispatch
    - Translation
    - Handling and exceptions
    - This is where a LOT of the work goes
- Abstractions
  - Separable architecture
  - Extensible constructs

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Challenge

- How to minimize complexity of individual objects?

- Three general approaches
  - Inheritance
  - Composition
  - Aggregation

Inheritance

- All concerns in one object/class
  - inherit / override them separately
  - works best with multiple inheritance
  - example: draggable_icon
    - inherit appearance from “icon” (output aspects only)
    - inherit behavior from “draggable” (input aspects only)

- From a pure language perspective, multiple inheritance rare
- Java uses the *interface* and *abstract class* concepts to implement

Composition

- Put together interactive objects at larger scale than interactors
- Container objects
  - e.g., row and column layout objects
- Containers can also add input & output behavior to things they contain

Aggregation

- Different concerns in separate objects
  - Treat collection as “the interactor”
  - Slice up Seeheim
  - General approach: design patterns
- Classic architecture: “model-view-controller” (MVC)
  - from Smalltalk 80
  - Also presentation-abstraction-control (PAC)

Before we Start…

- Swing is all Java.
- You should know about, and understand:
  - Classes / Objects
  - Method Overloading
  - Inheritance
  - Polymorphism
  - Interfaces
  - How to read the Java2 API Documents

2D interface programming toolkits

- Tcl/Tk
- Motif/UIL
- IDEs (e.g. VB, MSVC++/MFC)
- Java AWT – the beginnings
- Java JFC Swing (Java2 - JDK >= 1.2)
- JBuilder and other Java IDEs
- etc…

What is Swing?

- A part of The Java Foundation Classes
  - Swing
    - Look and feel
    - Accessibility
    - Java 2D (Java 2 onwards)
    - Drag and Drop
    - etc
- Can be used to build Standalone Apps as well as Servlets and Applets

Getting started with Swing (1)

- Compiling & running programs
  - Swing is standard in Java 2 (JDK >= 1.2)
  - Use:
    - 'javac <program.java>' && 'java <program>'
    - Or Eclipse

Getting started with Swing (3)

- Swing, like the rest of the Java API is subdivided into packages:
  - `javax.swing,   javax.accessibility, javax.swing.border …`
- At the start of your code - always
  - `import javax.swing;`
  - `import javax.swing.event;`
- Most Swing programs also need
  - `import java.awt.*;`
  - `import java.awt.event.*;`

Using Swing and AWT

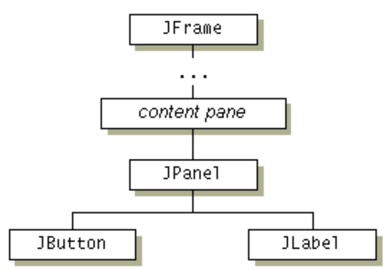
- ***Do not*** mix Swing and AWT components
  - Lightweight and heavyweight components cause side effects
- If you know AWT, put 'J' in front of everything
  - AWT: **`Button`**
  - Swing: **`JButton`**
- Swing does all that AWT does, but better and there's much more of it

A typical Swing program

- Consists of multiple parts
  - Containers
  - Components
  - Events
  - Graphics
  - (Threads)
- We will look at each in turn

A simple Swing program - Containers

- Containers



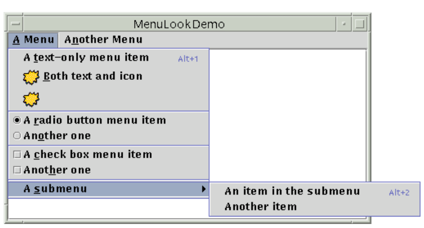
- JFrame, JDialog, JApplet

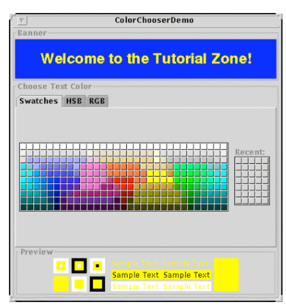
Remember this about Containers:

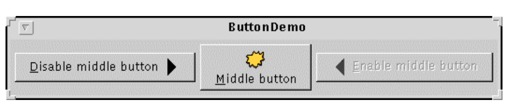
- The structure of containers is your design decision and should always be thought through in advance
  - particularly for managing components
  - nesting containers.
  - A component can only be in one container!
- Failure to do so usually either results in a messy interface, messy code or both.

A simple Swing program - Components

- Components







Components

- Components are added to Containers
- A Component can only live in one Container
- Components get added to the Container's *content pane*
  - In the case of `JFrame`, using the `setContentPane()` method.
  - Exception: we can add a menu bar to a Container

Remember this about Components:

- There are many components that make your job much easier.
- Often, you will be able to customise an existing Swing component to do a job for you, instead of having to start from scratch
  - Eg can extend (inherit from) the **`JButton`** class and 'paint' a new button over the top

The JComponent class

- All functions of interactors encapsulated in this base class

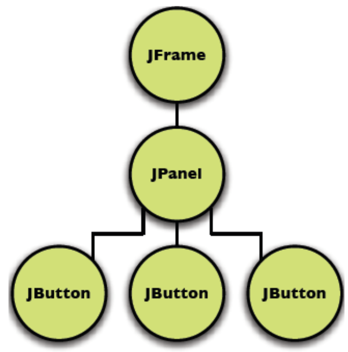
- Javax.swing.Jcomponent;
- Objects inherit from this class
- Methods for:
  - Hierarchy management
  - Geometry
  - Status
  - Layout
  - (re)drawing
  - picking

- In subclasses and other parts of the toolkit

- Input dispatch and handling
- Application interface
- Pluggable look and feel
- Undo support
- accessibility

Hierarchy Management

- Swing interfaces are trees of components
- To make something appear you must add it to the tree
  - In the right order
- Swing takes care of many of the details from there
  - Screen redraw
  - Input dispatch



Hierarchy Management

- Lots of methods for manipulating the tree
- add(), remove(), getComponent(), isAncestorOf(), getChildCount()

- Common mistake
  - If nothing shows up, make sure you have added it
  - setVisible() !

Geometry

- Every component maintains its own local geometry
- Bounding box:
  - getX(), getY(), getWidth(), getHeight()
  - 0,0 is at parent's upper left corner
  - setSize(), setLocation(), setBounds(), getSize(), getLocation(), getBounds()
- All drawing happens within the bounding box
  - Including output of children
- Drawing is relative to top-left corner
  - Each component has own coordinate system
  - Need to know dimensions of component

Object status

- Each component maintains information about its state
  - isEnabled(), setEnabled()
  - IsVisible(), setVisible()
- Lots of other methods of more limited importance

UI Software Organisation | IAT351 | January 18, 2008

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Each component handles

- Layout (coming later)
- Drawing
  - Component knows how to (re)create its appearance based on its current state
  - Responsible for painting 3 items in order
    1. Component
    2. Borders
    3. Children
  - `paintComponent(), paintBorder(), paintChildren()`
  - These are the only places to draw on the screen! BUT
  - Automatically called by Jcomponent's `paint` method, itself called by the Swing repaintManager (figures out damaged regions)

Damage(Change) Management

- Damage: areas of a component that need to be redrawn
  - Generic term
- Sometimes computed automagically by RepaintManager
  - Window overlap, resize
- Other times: you need to flag changes or damage yourself to tell the system that something in the internal state has changed and the onscreen image needs to be updated
  - E.g. changing the colour of a label
- Managing damage yourself
  - `Repaint(Rectangle r)`
  - `<componentName>.repaint();`
  - Puts the indicated area or component on the the internal queue of regions to be redrawn

Assignment 1

- Goal: learn how to use basic Swing components
- Familiarise yourself with toolkit
- Application: a simple photo album
- Use Jframes (windows) , panes, buttons and labels to build simple windowed tool
- Base of assignments 2 and 3
- We will develop examples in the tutorial

How to Learn Swing

- Don’t even try.
- Learn general framework principles and design styles.
- Then use the API reference, and Swing Tutorials to discover detailed usage of each component.

How to read Java Docs (1)

- Java 2 (1.5.0) API Reference available at:
  - http://java.sun.com/j2se/1.5.0/docs/api/

- Split into 3 Sections (html frames):
  - Top Left: Packages
  - Bottom Left: Classes in Packages
  - Main Frame: Information about selected Class

How to read Java Docs (2)

- General idea is find class, and examine main frame for information.

- Main frame pages split into sections:
  - Package hierarchy & implemented interfaces
  - Class Description, and links to more info
  - Nested Class Summary – Detail in separate page
  - Fields - 2 types Class (static) and instance, plus fields inherited from parent classes / interfaces
  - Constructor Summary
  - Method Summary & inherited methods from parents
  - Detailed info on all summary sections