

# **DESIGN METHODOLOGIES AND GRAPHICAL NOTATION**

- **Diagramming Notations**
- **Data Flow Analysis Methods**
- **Data Flow Diagrams**
- **Data Dictionary and Its Content**
- **Functional Analysis Methods**
- **Function Diagrams**
- **State Transition Diagrams**
- **Object Diagram Conventions**
- **Entity Relationship Diagrams**
- **Object Interaction Diagrams**
- **Booch Diagrams**
- **Design Methodologies**

# **DIAGRAMMING NOTATIONS**

**Many diagramming notations are used during both requirements analysis and design:**

- **Data Flow Diagrams**
- **Function Diagrams**
- **State Transition Diagrams**
- **Entity Relationship Diagrams**

**Other diagramming notations are intended specifically for design and are often language-specific. These notations are often used when the implementation language is Ada:**

- **Object Interaction Diagrams**
- **Booch Diagrams**

**The application of these and other diagramming notations is a part of an organization's software development process.**

# **DATA FLOW ANALYSIS METHODS**

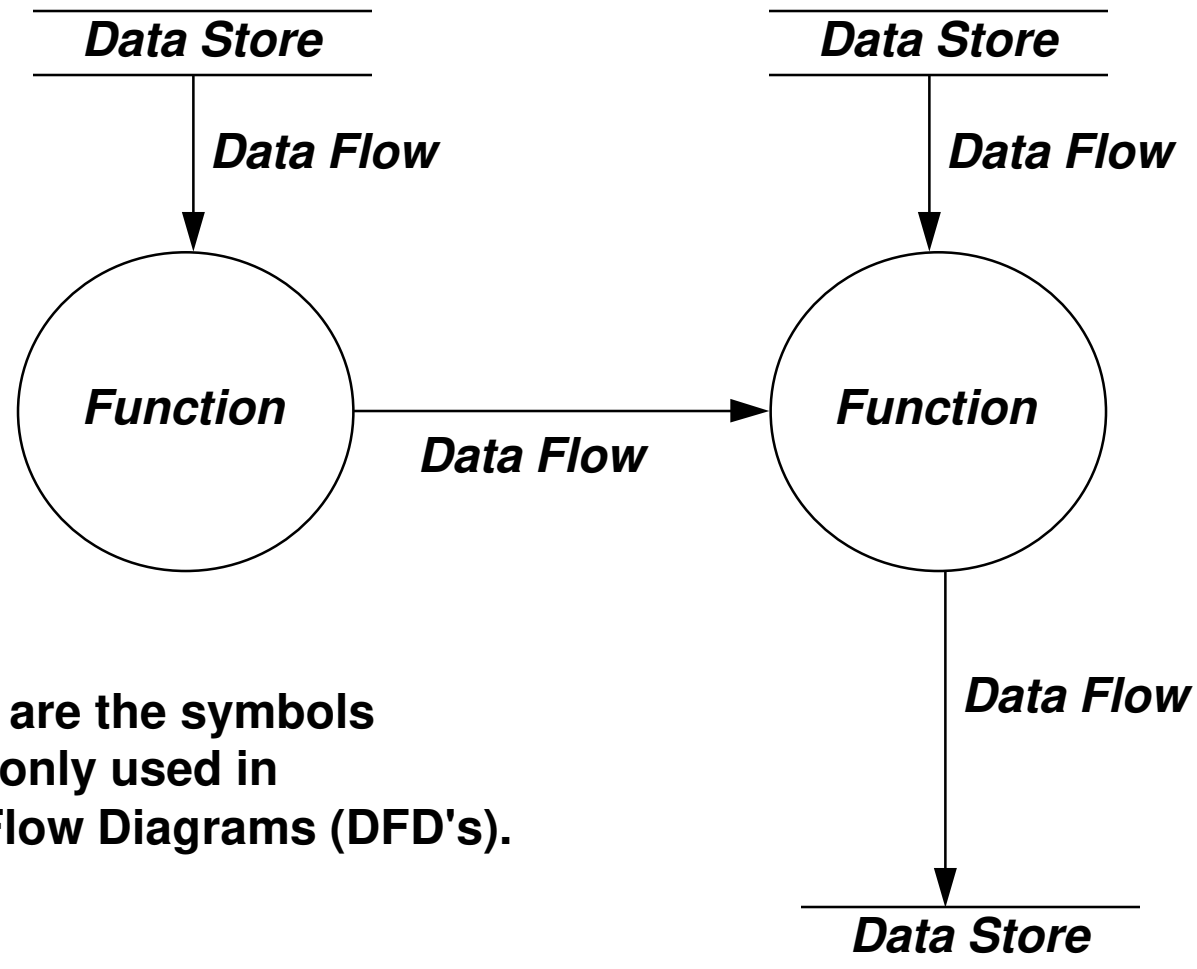
✓ **Data Flow Diagrams tell us:**

- **Data Sources and Sinks in the System**
- **Flow of Data in the System**
- **Functions which Transform the Data in the System**
- **Functions which cause Data Transactions in the System**

✓ **Data Dictionary tells us:**

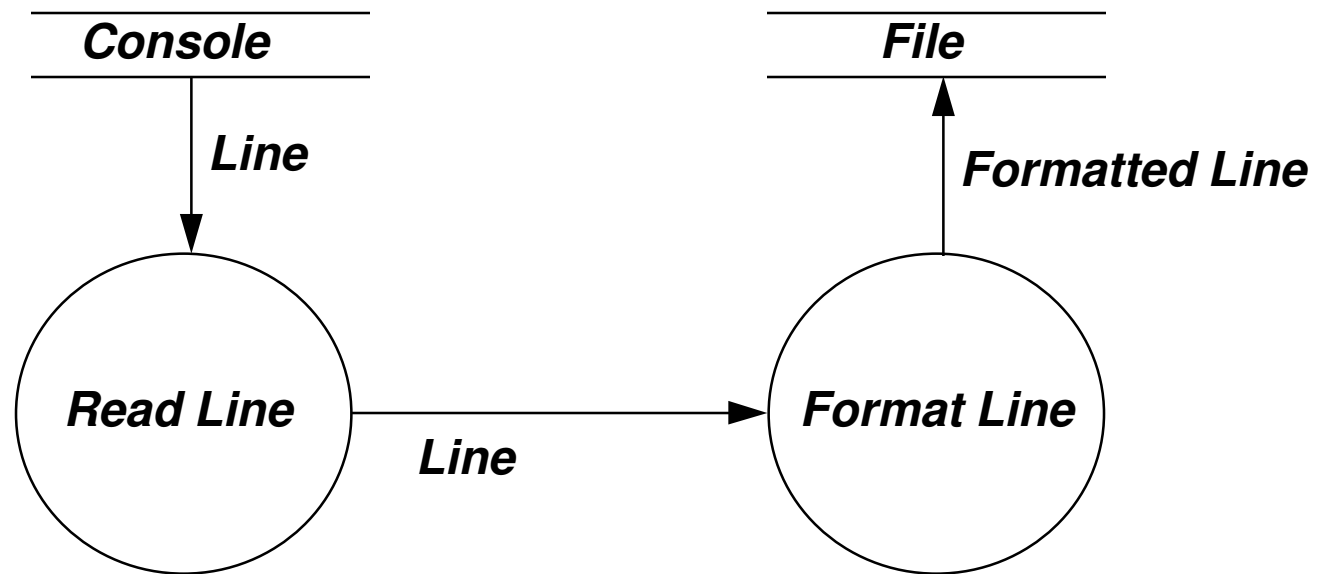
- **Attributes of the Data in the System**
- **Other Information about the Data in the System**

# **DATA FLOW DIAGRAMS**



**These are the symbols commonly used in Data Flow Diagrams (DFD's).**

# DATA FLOW DIAGRAMS EXAMPLE



# **DATA DICTIONARY AND ITS CONTENT**

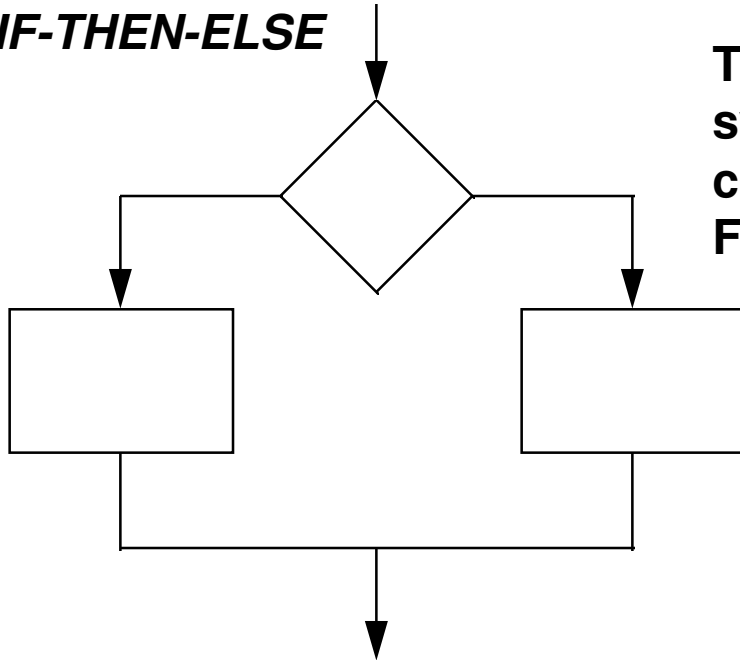
- **Each class of objects in the system and its attributes**
- **Each singular object (i.e., if placed into a class, the class would have only one instance) and its attributes**
- **Key constants and their attributes**
- **Subprogram parameters and their attributes**

# FUNCTIONAL ANALYSIS METHODS

- ✓ **Function Diagrams tell us:**
  - **Functions in the System**
  - **Sequence of Function Performance**
- ✓ **State Transition Diagrams (STD's) tell us:**
  - **States of the System**
  - **Relationships between States in the System**
  - **Events that Cause State Transitions in the System**
  - **Resulting Actions Performed in Response to these Events**

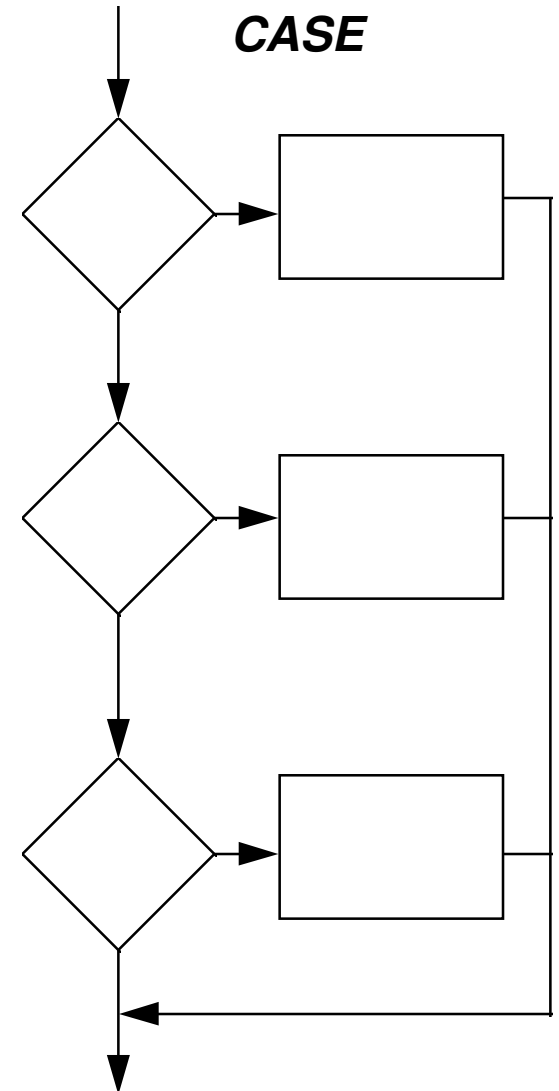
# FUNCTION DIAGRAMS

**IF-THEN-ELSE**

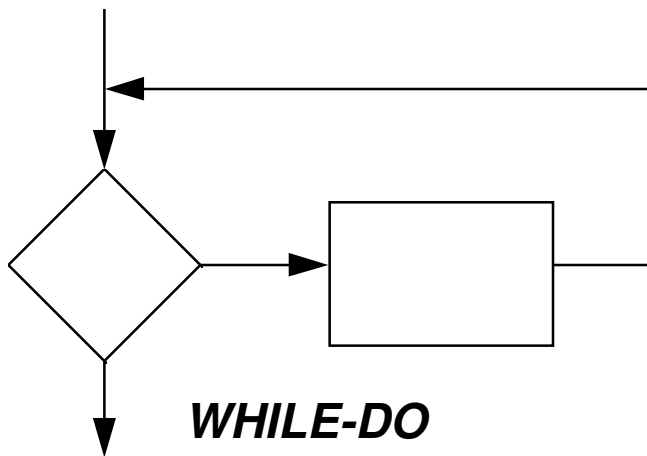


These are the symbol combinations commonly used in Function Diagrams.

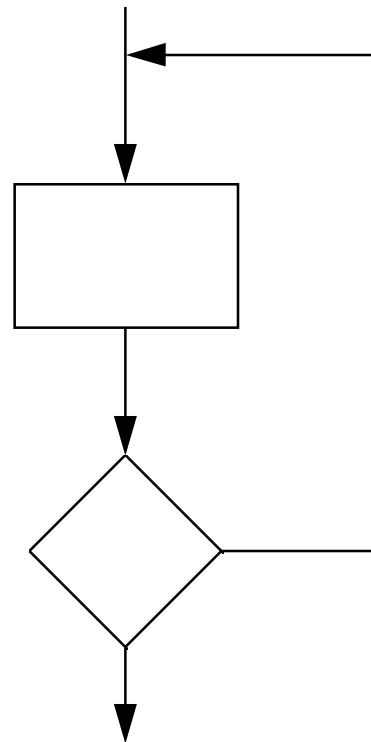
**CASE**



**WHILE-DO**

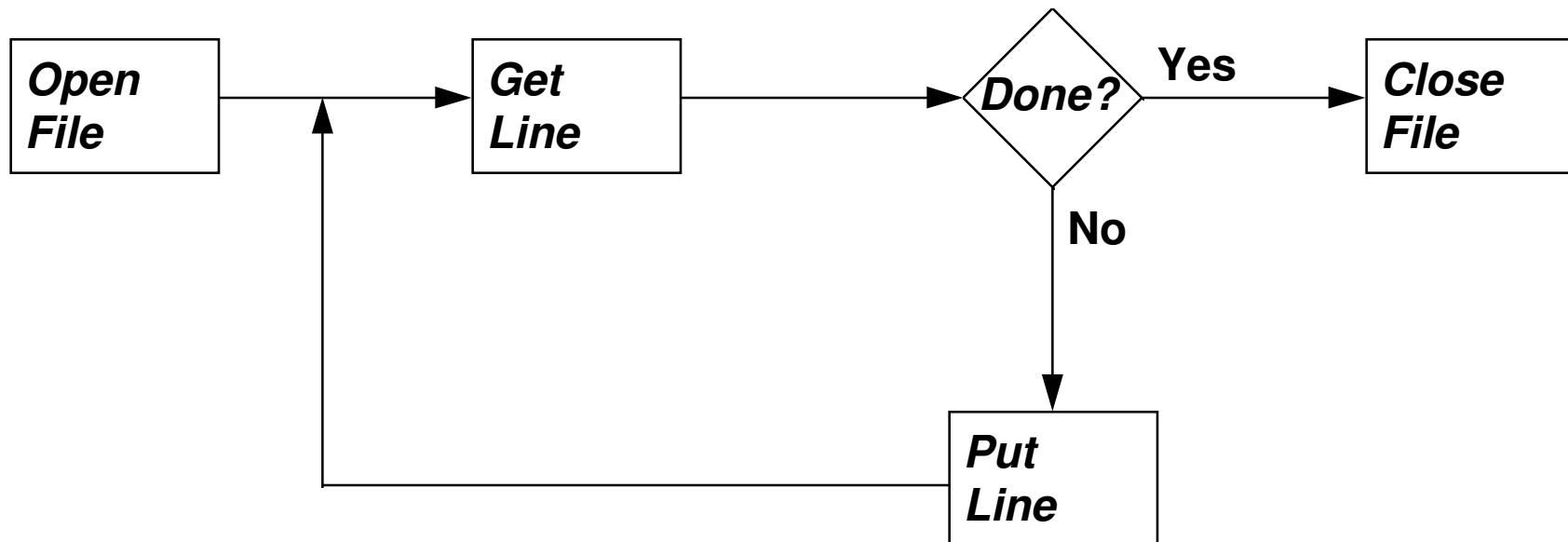


**REPEAT-UNTIL**

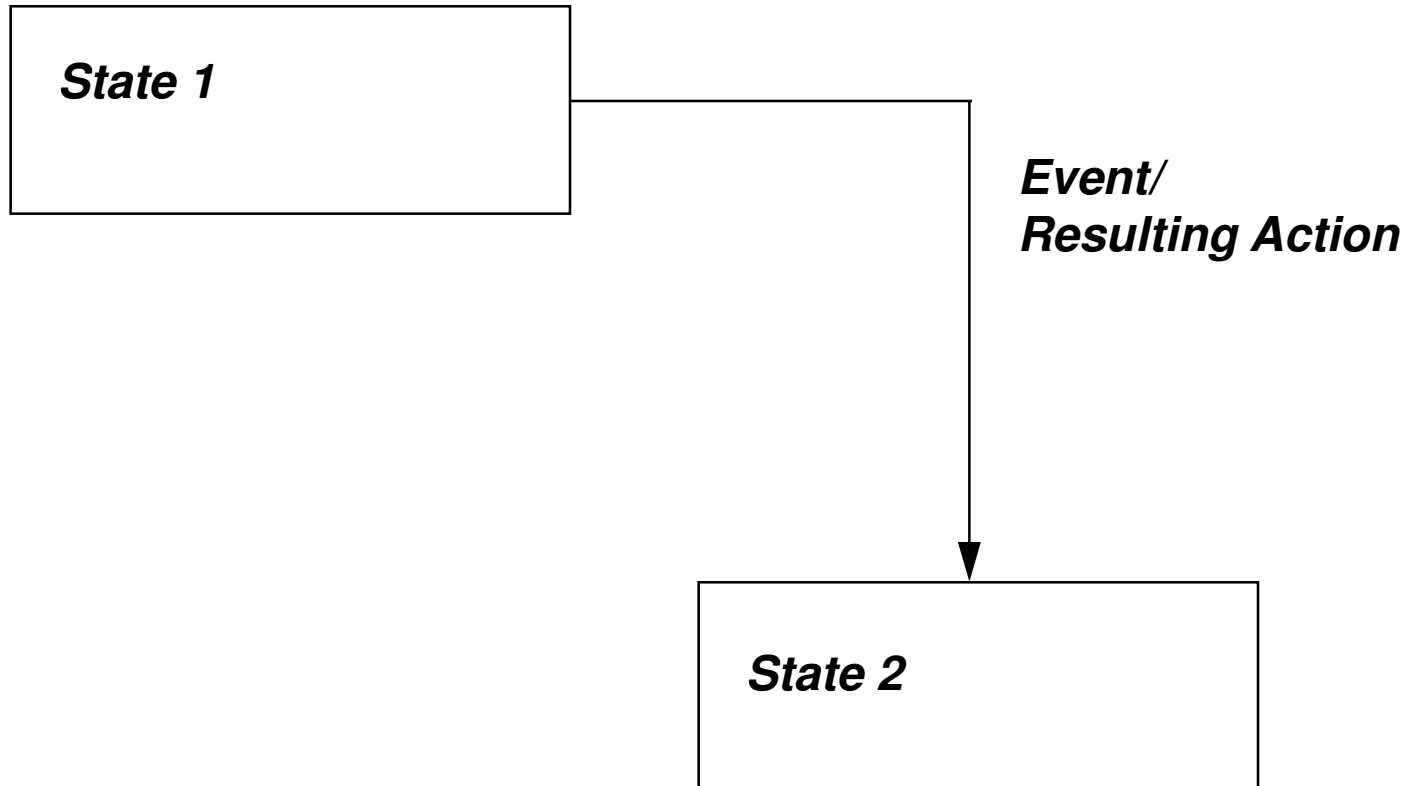




# FUNCTION DIAGRAMS EXAMPLE

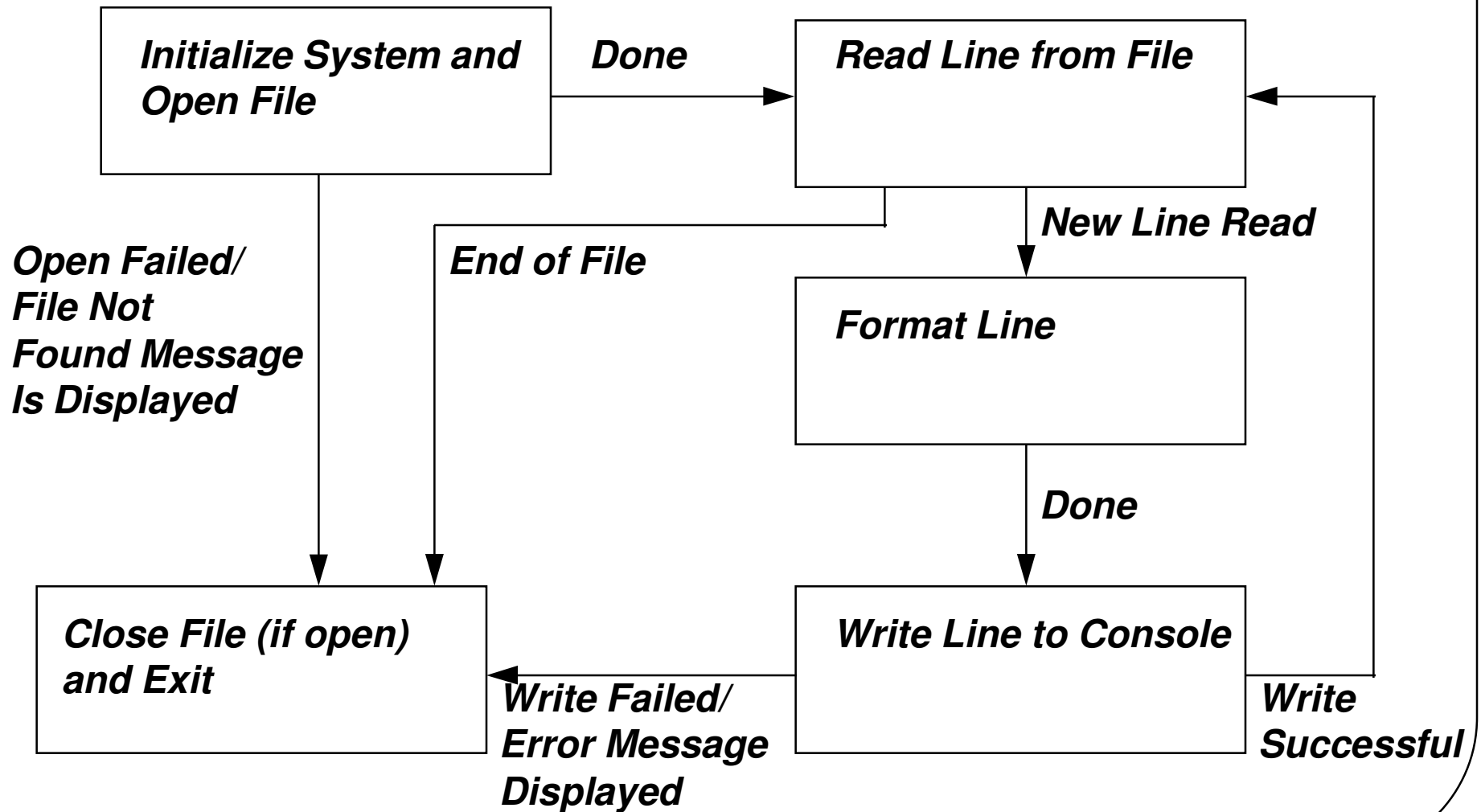


# STATE TRANSITION DIAGRAMS



These are the symbols  
commonly used in  
State Transition Diagrams (STD's).

# STATE TRANSITION DIAGRAMS EXAMPLE



# **OBJECT DIAGRAM CONVENTIONS**

✓ **Entity Relationship Diagrams (ERD's) tell us:**

- **Entities in the System**
- **Relationships between these Entities**

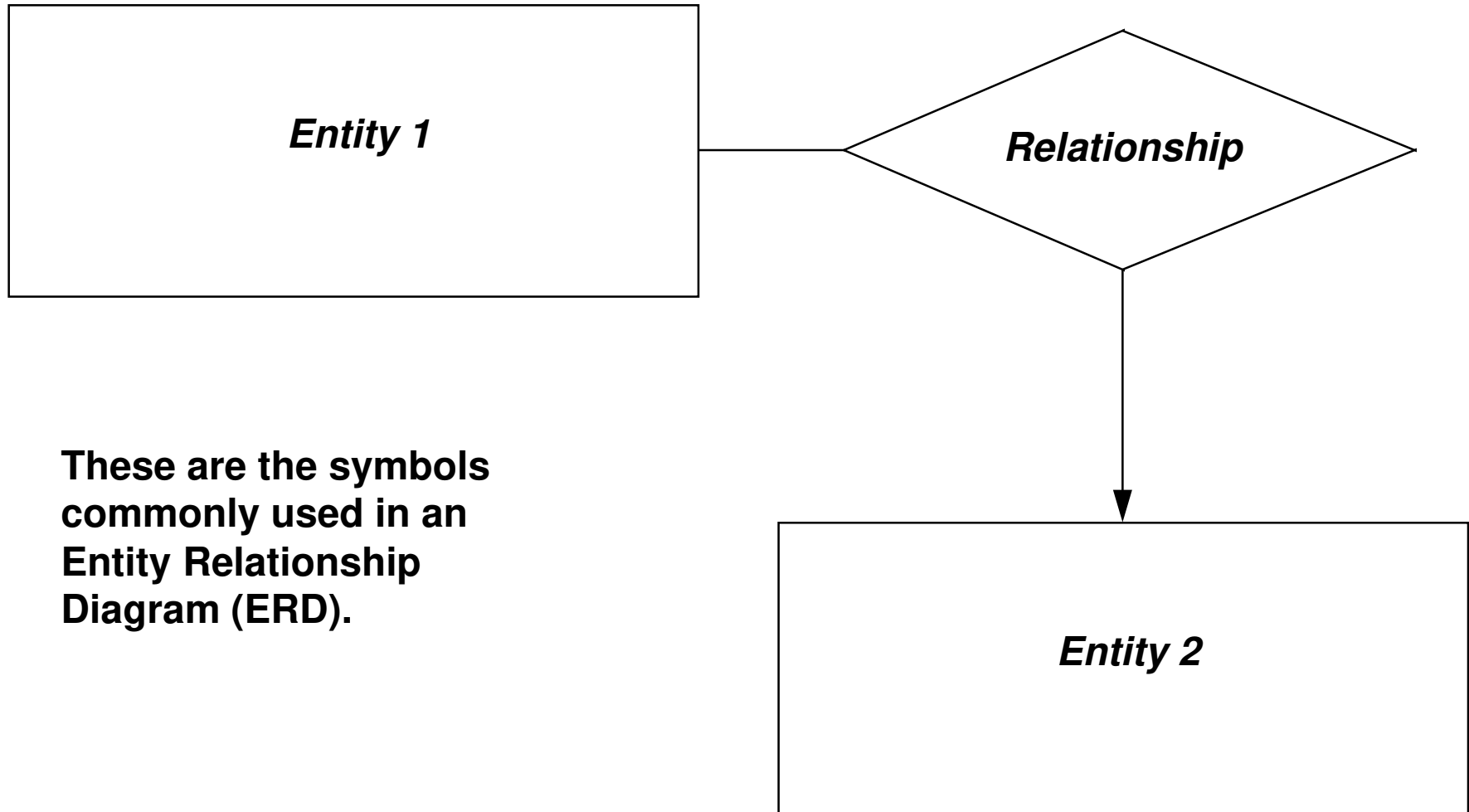
✓ **Object Interaction Diagrams tell us:**

- **Objects and Classes in the System**
- **Relationships between Objects**
- **Object Interfaces**
- **Data Flow between Objects**
- **Method Invocation**
- **Sequencing of Invocations (optional)**

✓ **Booch Diagrams tell us:**

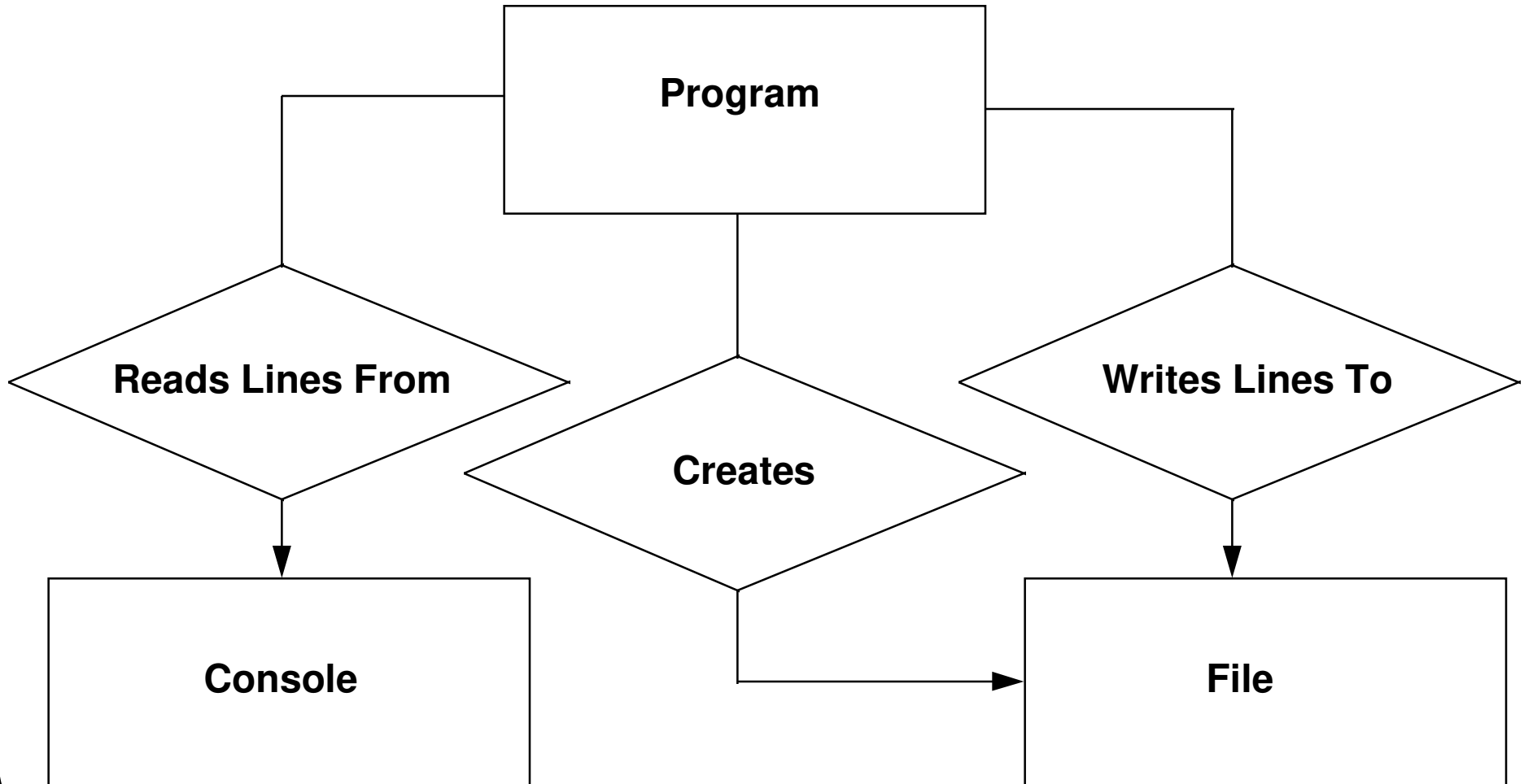
- **Dependency Relationships between Classes**

# ENTITY RELATIONSHIP DIAGRAMS



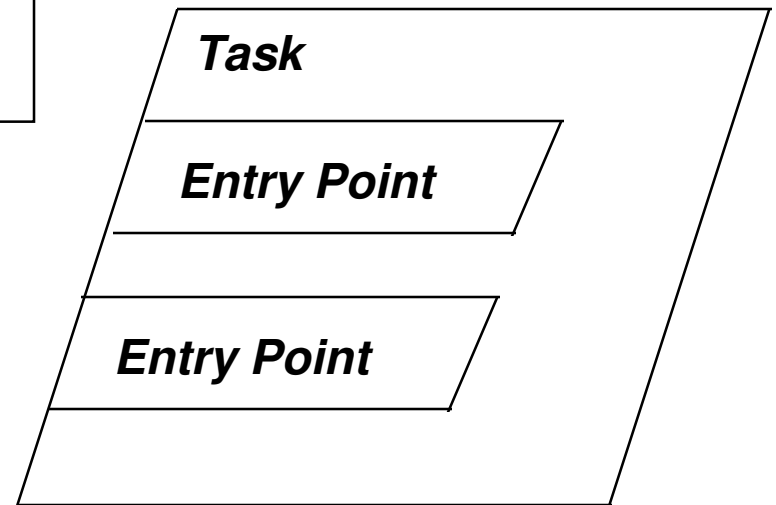
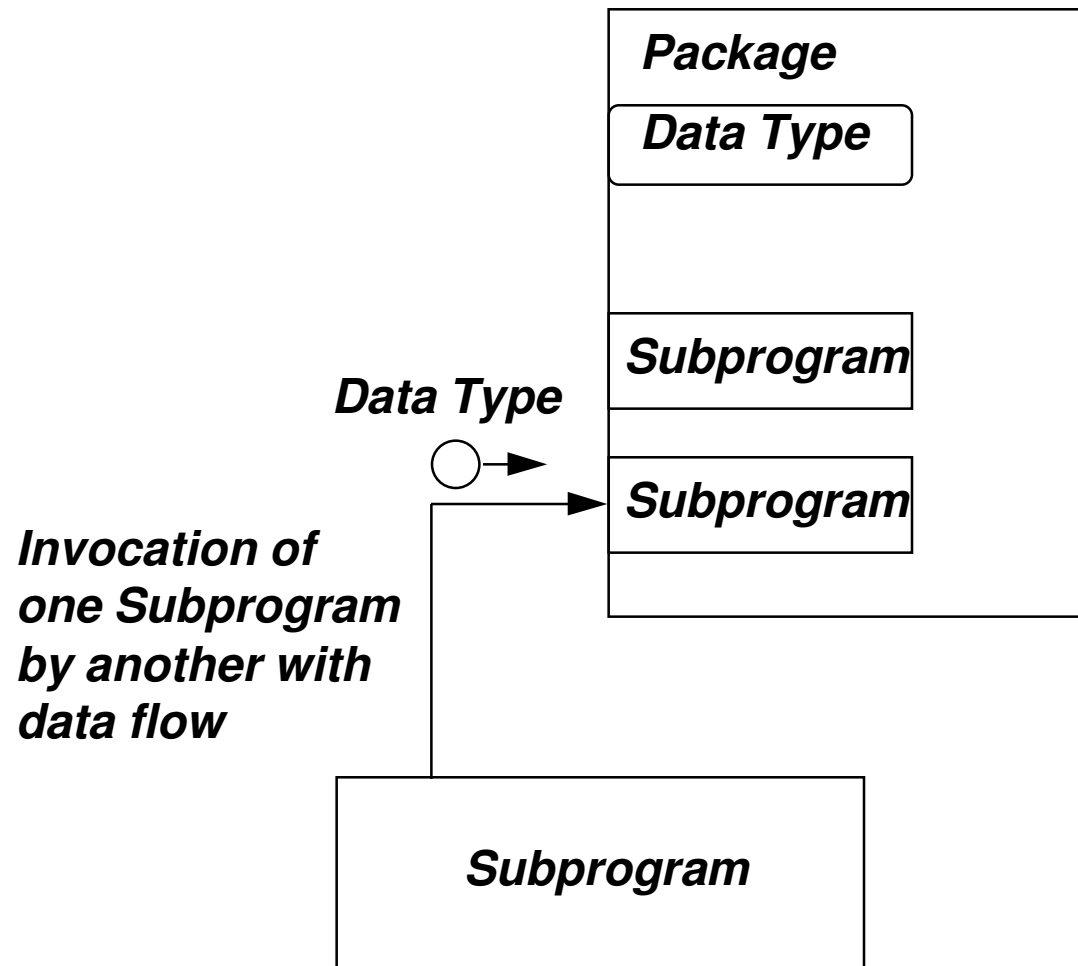
**These are the symbols commonly used in an Entity Relationship Diagram (ERD).**

# ENTITY RELATIONSHIP DIAGRAMS EXAMPLE

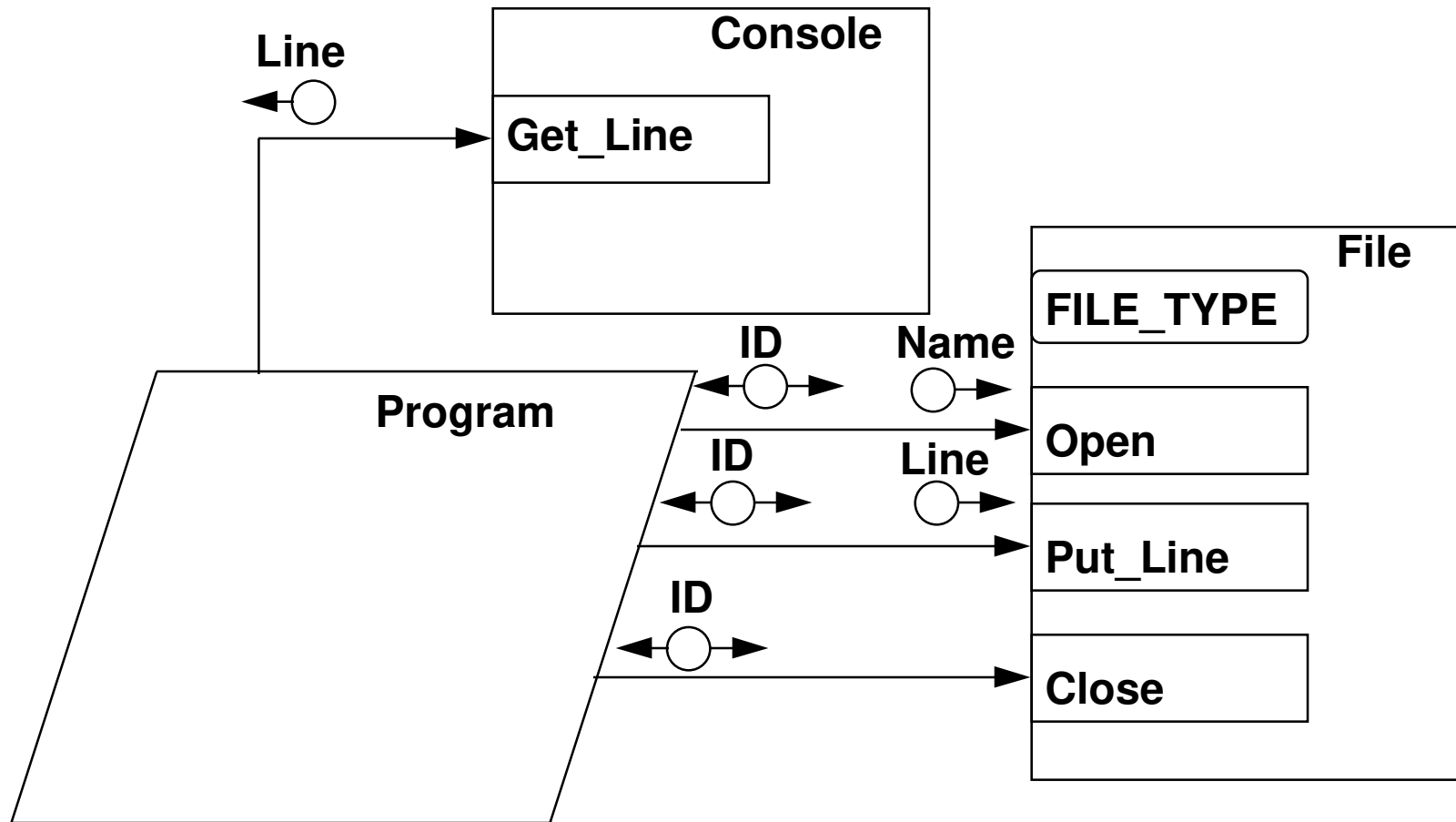


# OBJECT INTERACTION DIAGRAMS

These are the symbols commonly used in Object Interaction Diagrams (OID's).

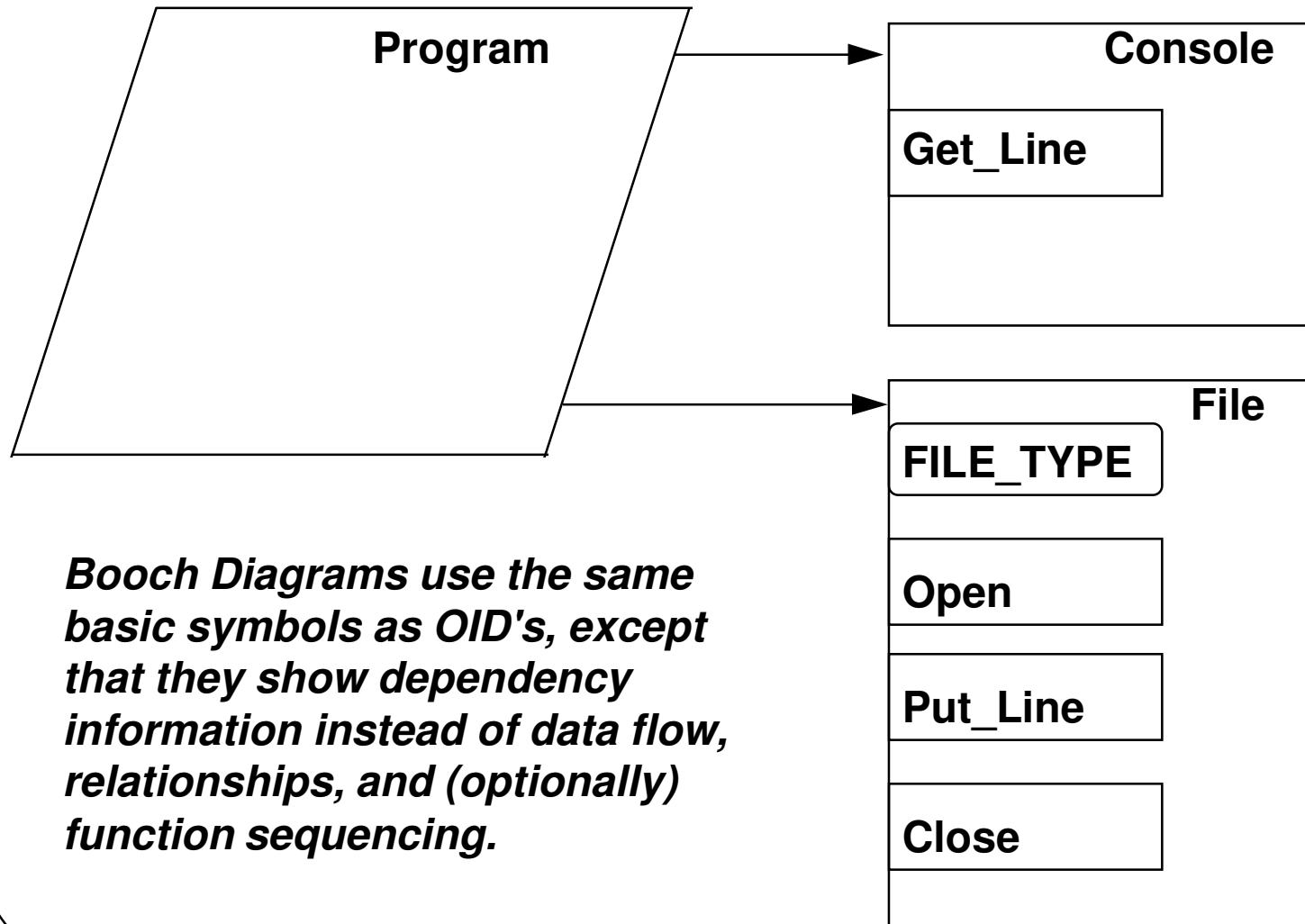


# OBJECT INTERACTION DIAGRAMS EXAMPLE





# BOOCH DIAGRAMS EXAMPLE (Only)



*Booch Diagrams use the same basic symbols as OOD's, except that they show dependency information instead of data flow, relationships, and (optionally) function sequencing.*

# **DESIGN METHODOLOGIES**

- ✓ **Data Flow-Oriented Design**
- ✓ **Data Structure-Oriented Design**
- ✓ **Object-Oriented Design**
- ✓ **Real-Time Design**

## **Note**

**The first three classes are heavily driven by  
the *Information Domain*.**

# Data Flow-Oriented Design

- Uses information flow characteristics to derive the program structure
- There are two design analysis techniques:
  - *Transform Analysis and Design* - the information flow exhibits distinct boundaries between incoming and outgoing data (i.e., input, processing, and output are the three key elements of the data flow)
  - *Transaction Analysis and Design* - an information item causes the flow to branch along a choice of paths
- Data Flow Diagrams (DFD's) are the common graphical means to represent the flow of data

# Data Flow-Oriented Design Transform Analysis and Design

## Design Steps:

- Review the fundamental system model
- Review and refine the DFD's for the software
- Determine the transform and transaction characteristics of the DFD's
- Isolate the transform center by specifying incoming and outgoing flows
- Perform "first-level factoring" - derive the mapping from the major parts of the DFD to a program structure
- Perform "second-level factoring" - map individual bubbles in the DFD into modules in the program structure
- Refine the above "first-cut" program structure - maximize cohesion, minimize coupling, and build a structure hierarchy

# Data Flow-Oriented Design Transaction Analysis and Design

## Design Steps:

- Review the fundamental system model
- Review and refine the DFD's for the software
- Determine the transform and transaction characteristics of the DFD's
- Isolate the transaction center and the flow characteristics of each action path
- Map the DFD into a software structure amenable to transaction processing
- Factor and refine the transaction structure and the structure of each action path
- Refine the above "first-cut" program structure - maximize cohesion, minimize coupling, and build a structure hierarchy

# Data Flow-Oriented Design Design Heuristics

- Minimize coupling and maximize cohesion
- Minimize fan-out and strive for fan-in as the depth increases
- Minimize side-effects; keep the scope of the effect of a module within the scope of control of that module
- Evaluate module interfaces to reduce complexity and redundancy; improve consistency of the module
- Define modules whose function is predictable and testable
- Strive for single-entry, single-exit modules
- Package software based on design constraints and portability requirements

# **Data Structure-Oriented Design**

- **Three key methods:**
  - ***Jackson System Development*** - concentrates on process modeling and control
  - ***Logical Construction of Programs (Warnier)*** - rigorous view of data structure and focus on detailed procedural design
  - ***Data Structured System Development (Orr)*** - incorporates data flow analysis with the Logical Construction of Programs and Jackson System Development (JSD to a lesser extent)
- **This is 1970's technology and is not covered in detail**

# **Object-Oriented Design (OOD)**

- **Concerns itself with creating a model of the real world**
- **Objects represent the information domain, and the operations associated with that information are grouped with the objects**
- **Messages (interfaces) provide a means by which operations are invoked**
- **Packaging of objects with their associated operations takes place - data and procedural abstractions are combined in a single program component called an *object* or a *package***
- **OOD representations are more prone than others to programming language dependency**



# Object-Oriented Design Definitions

- ***Object*** - a component of the real world that is mapped into the software domain or an information item
- ***Operations* or *Methods*** - processes which act on objects to transform their internal data structure or provide information on their internal data structures
- ***Message*** - a request to an object to perform one of its operations
- ***Class*** - a set of objects which share common characteristics
- ***Instance*** - an individual object of a class

# Object-Oriented Design Steps

- Identify the objects
- Identify the attributes of the objects
- Identify the operations that may be applied to the objects
- Establish the interfaces of the objects to the outside world (Ada package specifications may be used if Ada is the implementation language)
- Implement the objects (Ada package bodies may be used if Ada is the implementation language)
- Graphical representation may be employed; Booch Diagrams and Object Interaction Diagrams are the recommended diagramming notations

# Real-Time Design

- **Encompasses all aspects of conventional software design while simultaneously introducing timing and sizing constraints; these constraints must be satisfied by the code**
- **All classes of design (architectural, procedural, and data) become more complex due to the response time required by the real-world constraints**
- **Mathematical modeling and simulation are common tools used for real-time design**

# **Real-Time System Concerns**

- **Interrupt handling and context switching**
- **Response time**
- **Data transfer rate**
- **CPU and system throughput**
- **Resource allocation and priority handling**
- **Task synchronization and intertask communication**