Rational's Unified Modelling Language (UML)

Unified Software Development Process (USDP)

Initial Goal: unify the approaches advocated by three major contributors to OO-Design:

- Grady Booch
- Jim Rumbaugh
- Ivar Jacobson (3 Amigos)

Success has far exceeded this goal because:

- Political: UML developed by a consortium led by three leaders in OOA/OOD; wide acceptance among software professionals; respect and confidence by majority of software industry.

- Marketing: Submitting UML specification to a standardization process within Object Management Group (OMG). OMG has over 900 member organizations; UML is perceived as an open, widely supported standard.

- Technical: Concentrated on a standard modelling language, not a standard modelling method; provides common notation.

What is defined?

- For end users, specific notations in which their designs are described (UML notation guide).

- Tool implementations supporting the UML notation will work with a detailed description of UML syntax and semantics.

UML: Different views of the underlying model expressed as graphical diagrams:

1) Use-case diagram: early in life cycle

2) Class diagram: captures vocabulary of system

3) Behavior Diagrams:

   - State diagram: captures event-based behavior; changes of state and model elements

   - Activity diagram: captures business workflows; restricted form of state diagram, where external events and asynchronous behavior are ignored; concentrates on internal behavior

- Sequence diagram: captures time-related behavior; can help optimize message traffic through the system.

- Collaboration diagram: captures message-oriented behavior of systems; can see interactions between pieces; more abstract view of data flow.

- Package diagram: captures descriptions of modelling elements into packages, and dependencies between packages.

4) Implementation diagrams:

- Component diagrams: captures the physical structure of the implementation of a system (Arch).

- Deployment diagrams: captures the physical topology of the system.

The attraction of many OO-analysis methods is that they encourage rapid design and deployment, facilitating an iterative and incremental approach to design.

Use Case Diagrams

A *use case diagram* describes the behavior of a system from a user's standpoint.

It includes actors, a system, and use cases.

An *actor* represents a role of a person or thing that interacts with the system.

A *use case* is a specific kind of system use.

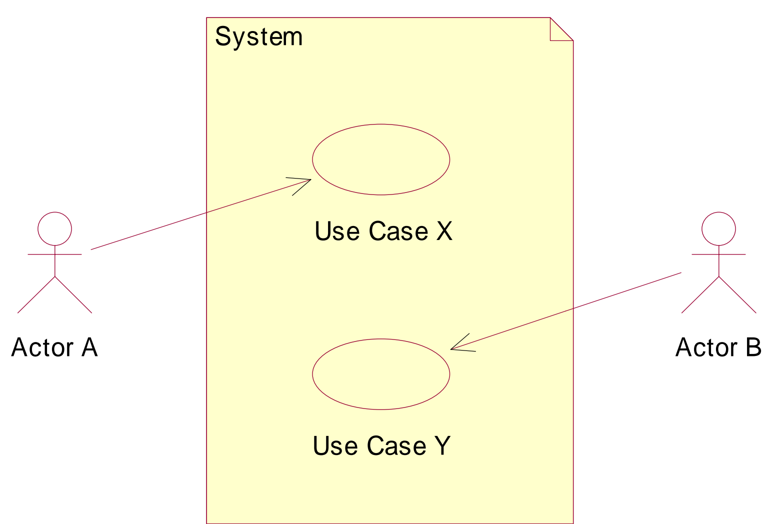
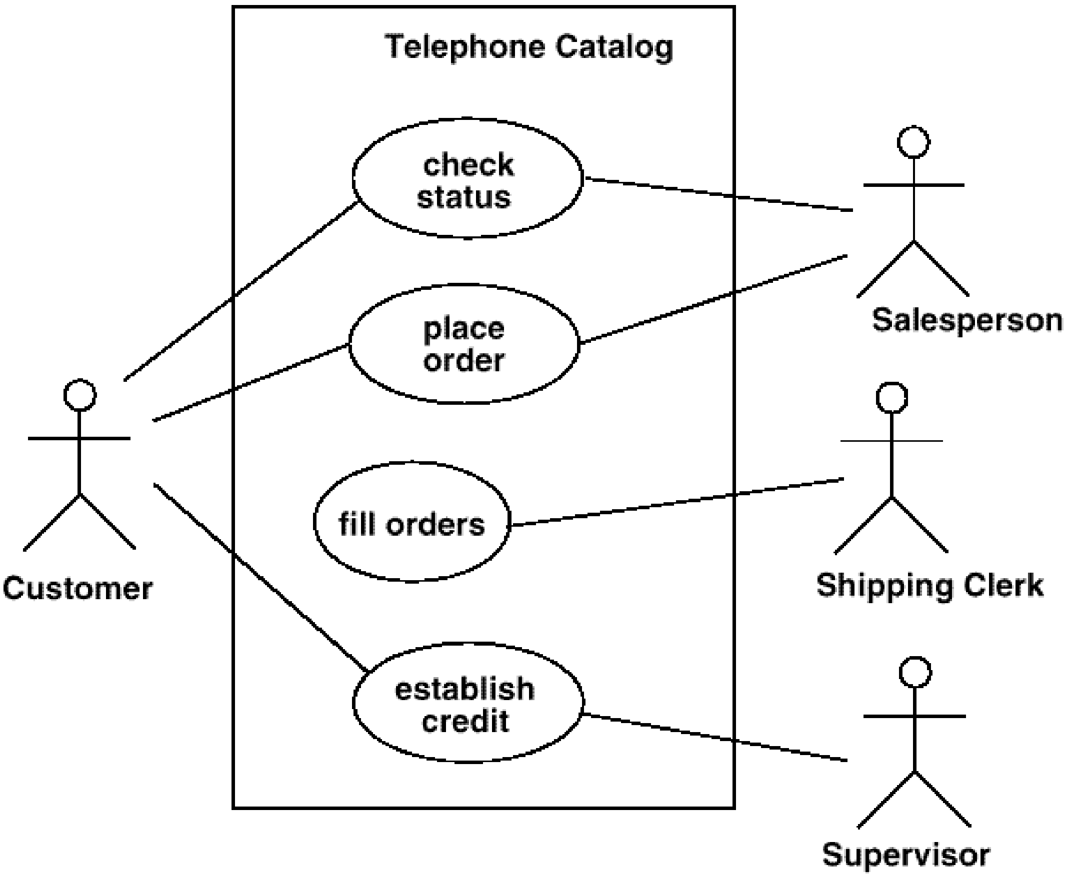


Figure 33. Use case diagram

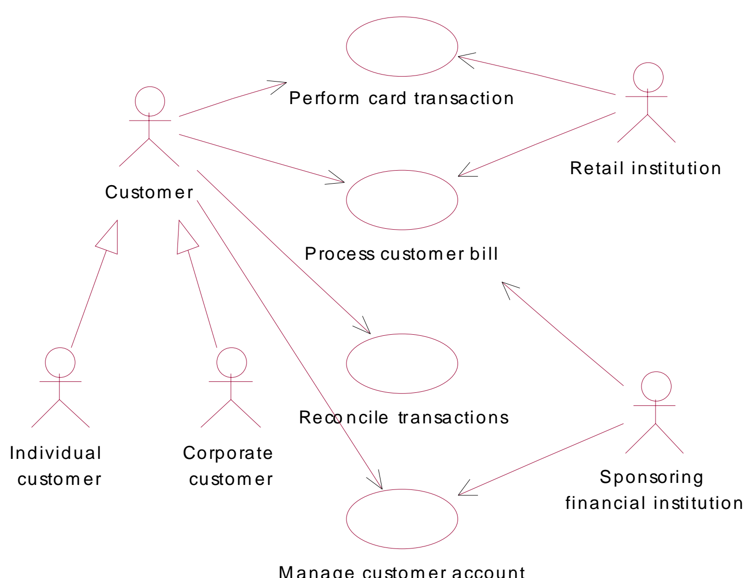


Four principal categories of actors:

- *Principal actors* — people who use the main system functions.
- *Secondary actors* — people that perform administrative and maintenance tasks.
- *External hardware*.
- *Other systems*.

Relationships between use cases:

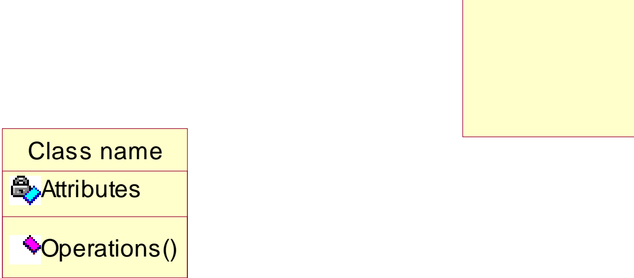
- *Communicates* — participation of actor.
- *Uses* — instance of source use case includes behavior desribed by target.
- *Extends* — source use case extends behavior of destination use case.



(From *The Unified Modelling Language User's Guide* by Booch, *et al.*

Classes

An object class is represented in UML by a rectangle with three compartments:



Attributes may be suppressed:

UML defines three visibility levels for attributes and operations:

- *public* — the element is visible to all clients of the class
- *protected* — the element is visible to subclasses of the class
- *private* — the element is visible only to the class

In standard UML, these levels are indicated by character annotations:

NewClass

- privateAttribute
protectedAttribute
+ publicAttribute

- privateOperation()
protectedOperation()
+ publicOperation()

Relationships Between Classes

Associations are relationships (of any kind) between two or more classes.

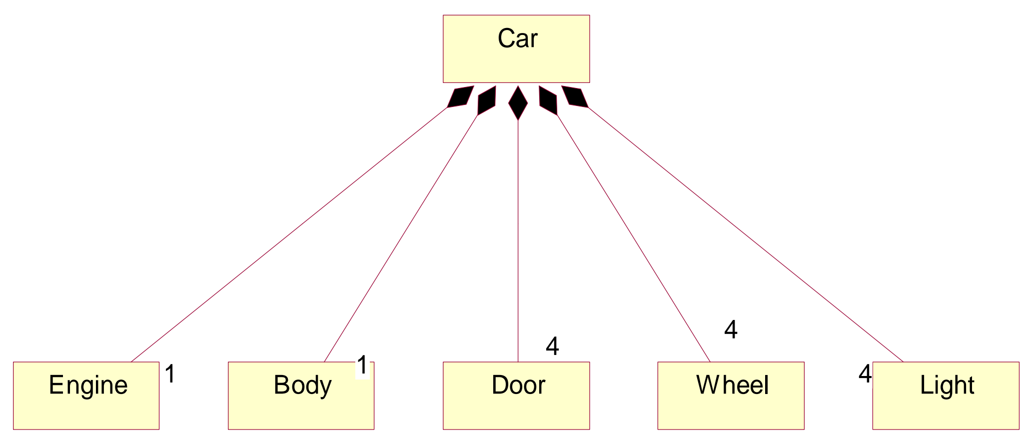
Links are instances of associations.

They are represented by lines:

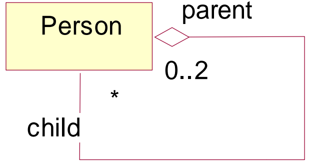
Associations can be named.

Classes can have *roles* in an association.

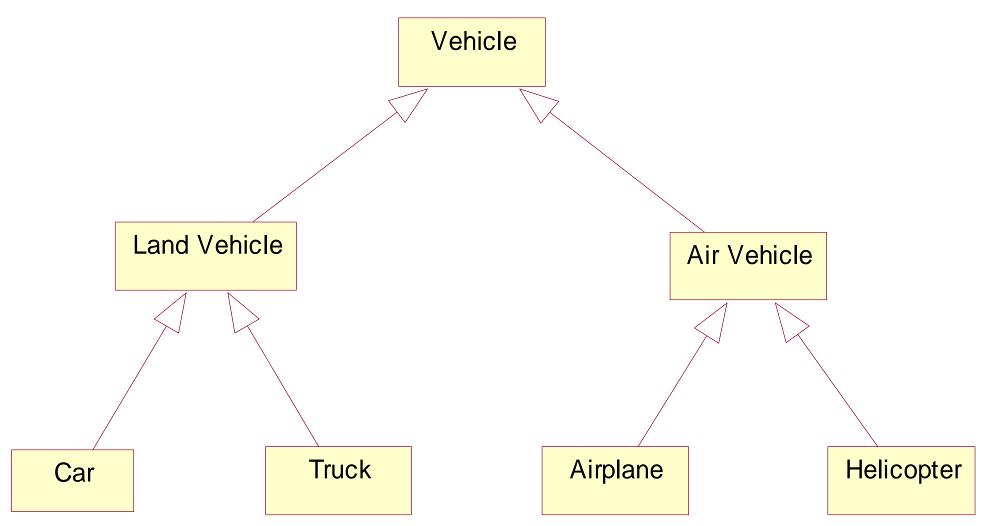
Composition is an association indicating that object of one class contain objects of another.



Aggregation is a somewhat weaker association, which represents a transitive, asymmetric, and possibly reflexive relationship.



Class-subclass relationships are represented in UML using the *generalization* relationship, e.g.,



Generalization is a transitive but irreflexive relationship.

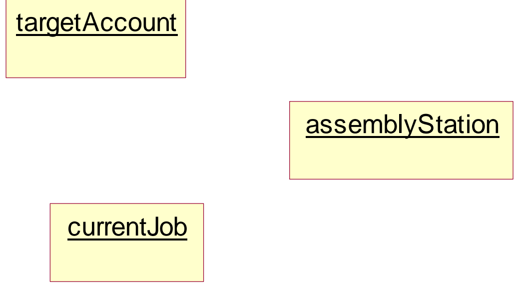
Behavoral Models/Diagrams

Show Dynamic Behavior

Objects

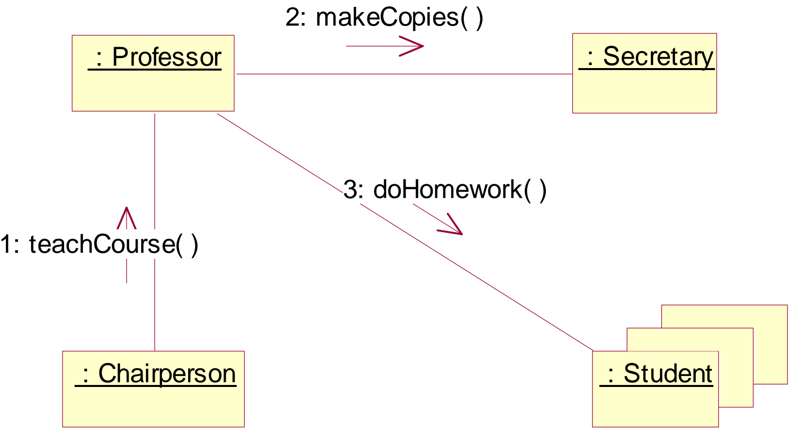
In UML, objects are represented by rectangles.

The name of the object is underlined.



Interaction Diagrams

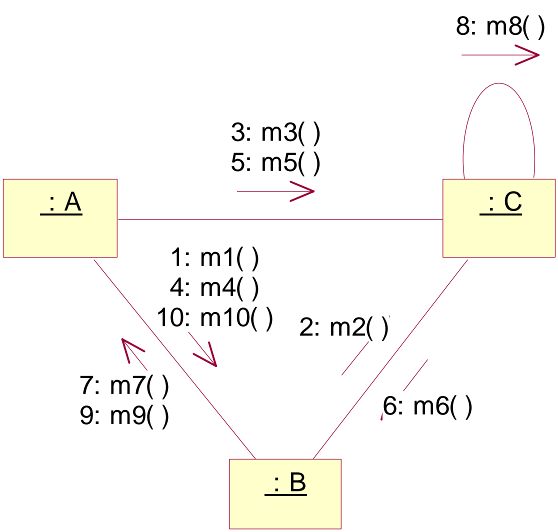
Interactions between objects are represented by lines called *links*.



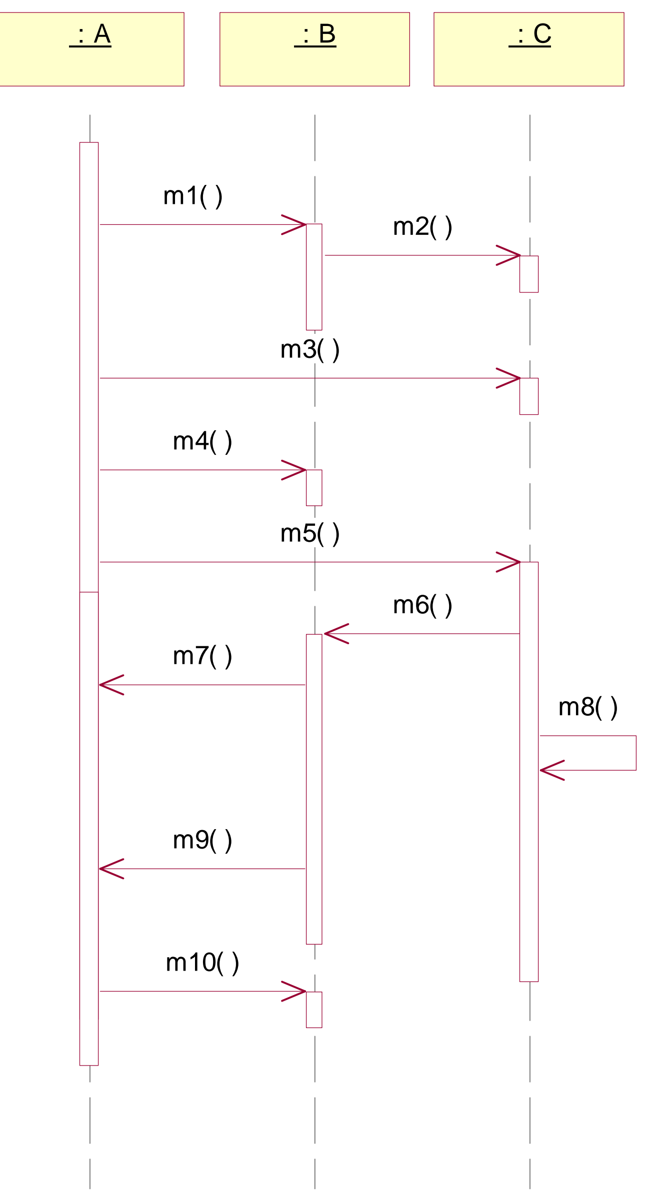
Messages follow links and are represented by arrows.

Collaboration diagrams like the one above represent interactions between objects.

Collaboration diagrams can be difficult to understand if they contain many messages.



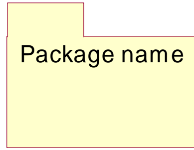
Sequence diagrams use timelines to show the sequencing of messages.



Packages

Packages are used for grouping model elements.

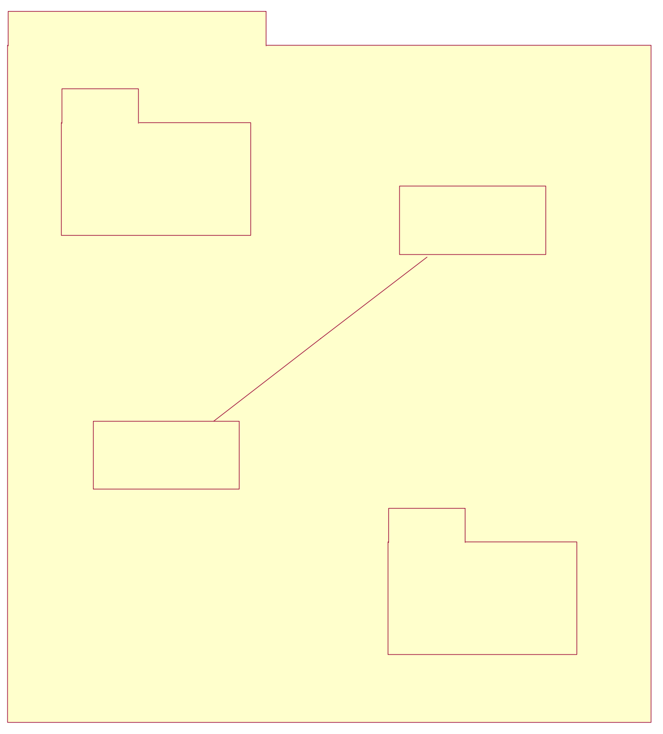
A package is denoted by a folder:



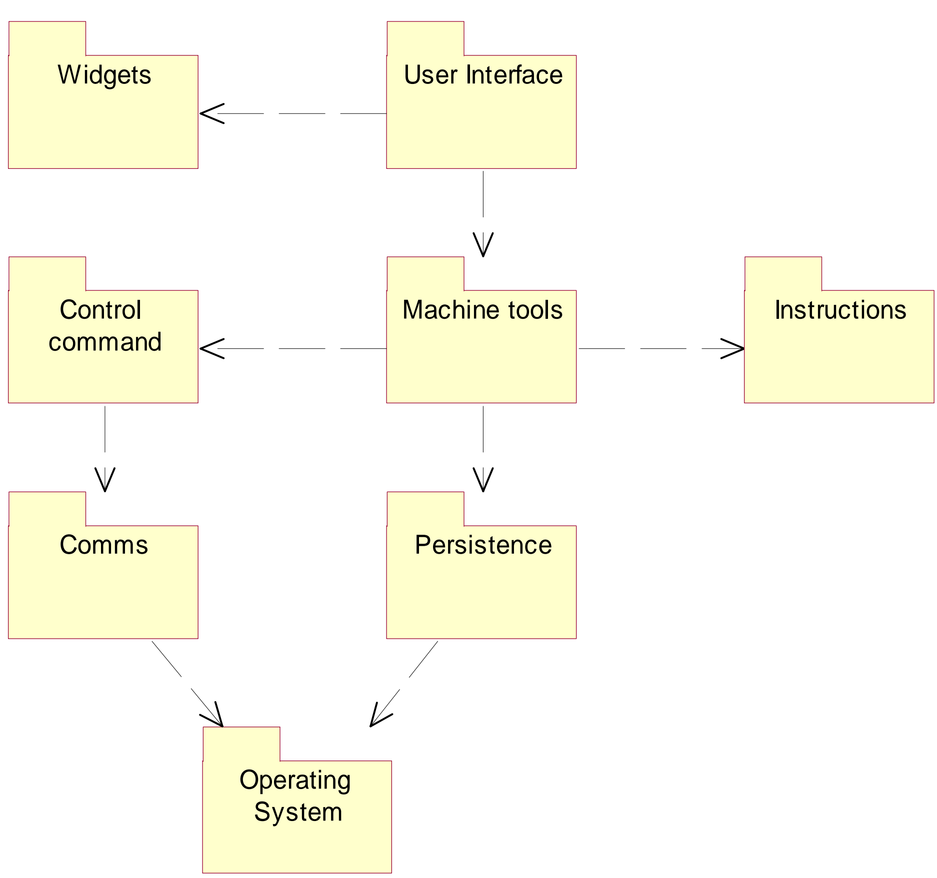
Each package corresponds to a subset of the model.

It may contain classes, objects, relationships, components, nodes, and associated diagrams.

A package may contain other packages, e.g.,



The general system layout is expressed by the hierarchy of packages and by dependencies between packages, e.g.,



Implementation Diagrams

Patterns:

Can help and teach novice designers: a "model answer". In UML, patterns can be used to explain the design and its suitability.

Stereotypes:

UML has an extension that uses stereotypes. These can be used with any diagram to extend its meaning. A number of predefined UML class stereotypes can be used for events, exceptions and interfaces (also can be user-defined).

Limitations of UML for Enterprise Scale Component Modeling

- To model interfaces and components, user must use extensions in UML, such as sereotypes and naming conventions; so no first class example in UML tools
- Difficult to describe the dynamic behavior defining how multiple components collaborate to satisfy a required behavior (use cases, sequence diagrams just given)
- Interface descriptions must be given in detail to allow component producer and consumer to work independently; UML does not support this; no pre- or post-condition pairs
- UML provides no support for any component modeling method.

- **Where UML Diagrams Fall Short**
- Best way to see not always with a diagram
- Sorting algorithms are not shown in a sequence diagram
- A sequence of messages doesn't illustrate side effects
- So what happens when an object is added to a hash table or a buffer overflows
- Without special annotation of diagrams, what conditions cause branching, iteration or successful completion of the algorithm?
- Algorithm details may be better described in text, pseudocode, code, BNF, FSM models, decision tables or other representations.

Various Architectures and UML

UML and Architecture

- Need for more precise and detailed descriptions of design than these general architectures can provide.

- UML provides this as a standard graphical modeling language.

- Provides a vocabulary for describing classes, objects, roles, interface, collaborations and other design elements.

- But UML is more than a graphical notation, as a well-defined semantics lies behind each symbol.

- This means you can specify a UML model using one design tool, and another tool can interpret that model unambiguously.

Applying Double Dispatch to a Common Children's Game

- To implement the game "Rock, Scissors, Paper", find code to determine whether one object beats another.
- Game has 9 outcomes based on the three kinds of objects.

Initial Solution

- Use **case** or **switch** governed by the data operated on.

- Following Java program is not a very good solution.

- Receiver needs to know too much about the argument.

- One of these nested conditionals is needed in each of the three classes.

- If new kinds of objects are added to the game, all classes would have to be modified.

A Better Solution to the Game

- Use **double dispatch** pattern to avoid touching any working methods.

- Note rock or paper does not need to know what kind of object is being compared.

- Extending the game to another object simply requires adding another method in each existing class, and creating a new class that implements the new Game-object interface.

Frameworks

- A **framework** is a general design to solve a software problem.

- Unlike a pattern, which is an idea of how to solve a familiar problem, a framework provides a library of classes that developers can tailor or extend to fit a particular situation.

- The success of a framework depends on how useful it is to developers and how easily it can be tailored to their needs.

Examples of Frameworks

1) GUI: Java Swing framework offers a set of features useful for building an interactive user interface.

2) Simulation: The early Smalltalk 80 PL included a framework for building discrete event simulations.

3) Prog. Environments: Eclipse IDE (integrated development environment) has a plug-in architecture that lets tool providers supply different compilers, refactoring tools and debuggers.

4) Web applications: MS .NET framework is a unified set of tools for building distributed applications, including frameworks for building user interfaces, performing transactions and concurrences, interoperating between platforms and building web services.

Framework Advantages

- Efficiency: Less design and coding
- Richness: Domain expertise captured in the framework
- Consistency: Developers become familiar with the approach imposed by the framework
- Predictability: A framework has undergone several iterations of development and testing.

Costs & Disadvantages of Frameworks

- Complexity: steep learning curve
- If you have a hammer, everything looks like a nail; frameworks require a specific approach for solving the problem
- Performance: trades flexibility and reusability for performance.

(Documenting) Flexible Designs
Templates and Hooks

- UML-F: Fontoura, Pree & Bernhard extend UML notation.

- complete c or incomplete . . .

- Can annotate methods with an explanation of their intent & implementation by specifying:

  - abstract and needs to be overridden by subclasses
  - inherited and not redefined
  - newly defined or completely redefined by a class
  - redefined but uses behavior from superclass

  - shows presence of **hooks.**