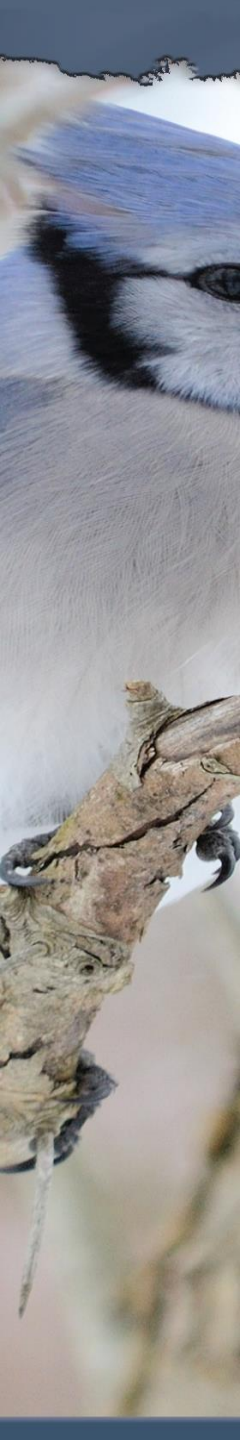
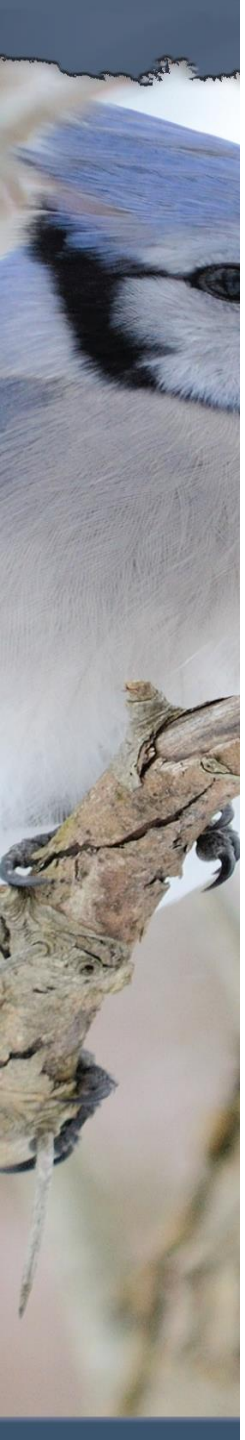
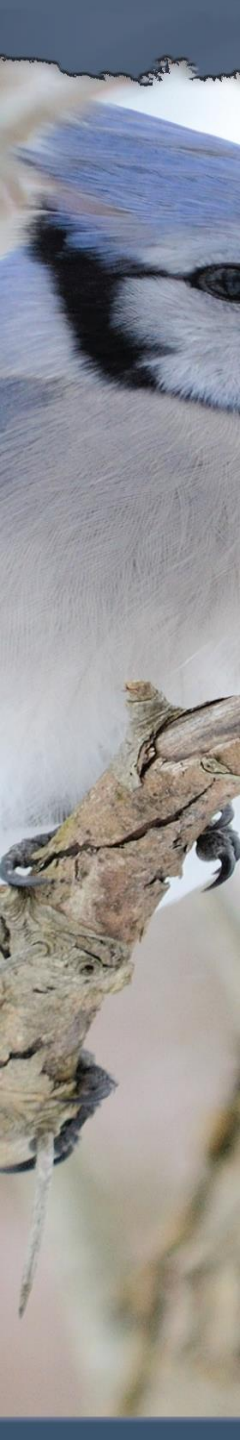


Designing applications



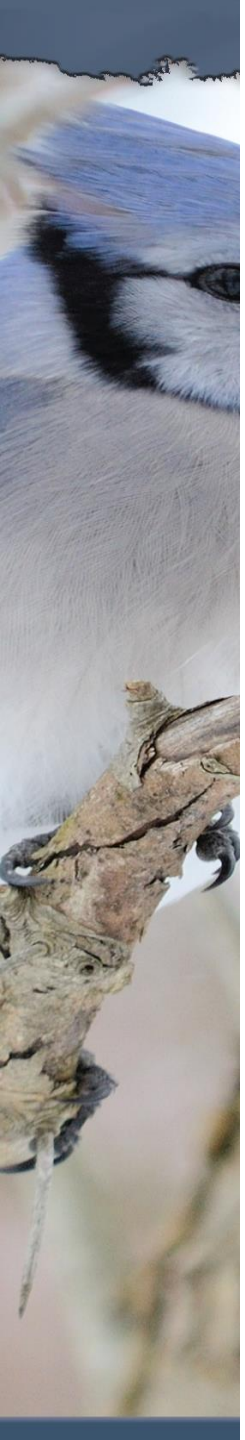
Main concepts to be covered

- Discovering classes
- CRC cards
- Designing interfaces
- Patterns



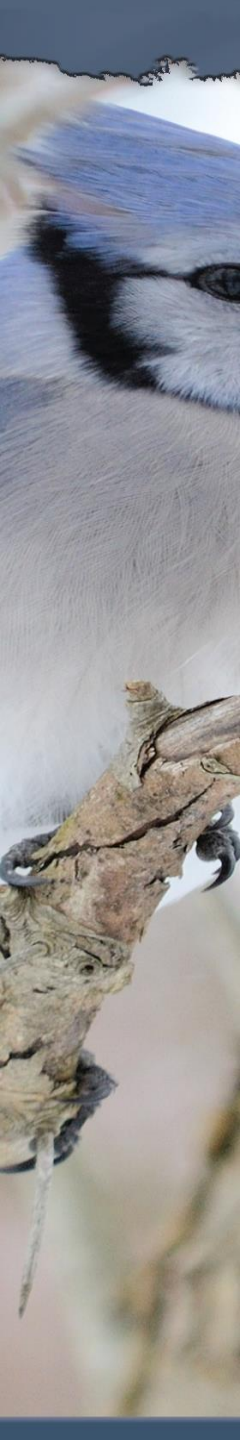
Analysis and design

- A large and complex area.
- The verb/noun method is suitable for relatively small problems.
- CRC cards support the design process.
  - Class
  - Responsibilities
  - Collaborators



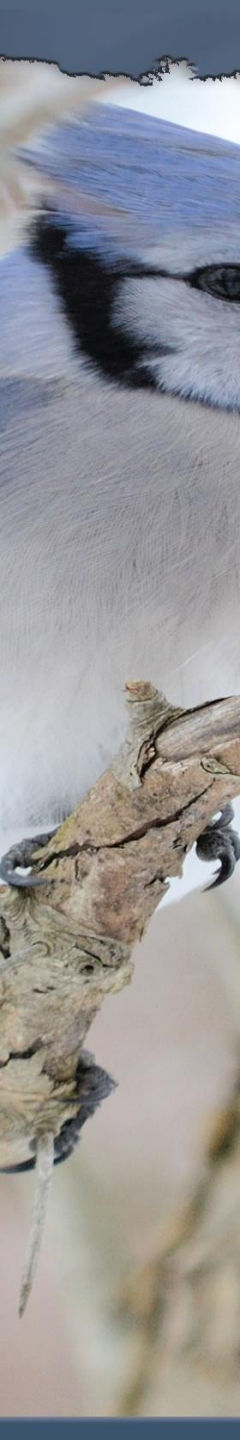
The verb/noun method

- The nouns in a description refer to ‘things’.
  - A source of classes and objects.
- The verbs refer to actions.
  - A source of interactions between objects.
  - Actions are behavior, and hence methods.



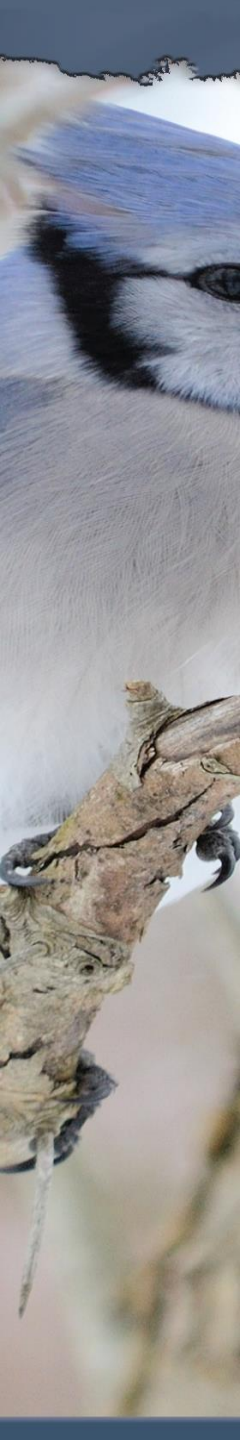
A problem description (1)

- The cinema booking system should store seat bookings for multiple theaters.
- Each theater has seats arranged in rows.
- Customers can reserve seats and are given a row number and seat number.
- They may request bookings of several adjoining seats.



A problem description (2)

- Each booking is for a particular show (i.e., the
- screening of a given movie at a certain time).
- Shows are at an assigned date and time, and scheduled
- in a theater where they are screened.
- The system stores the customer’s phone number.



Nouns and verbs

Cinema booking system
Stores (seat bookings)
Stores (phone number)

Theater
Has (seats)

Movie

Customer
Reserves (seats)
Is given (row number, seat number)
Requests (seat booking)

Time

Date

Seat booking

Show
Is scheduled (in theater)

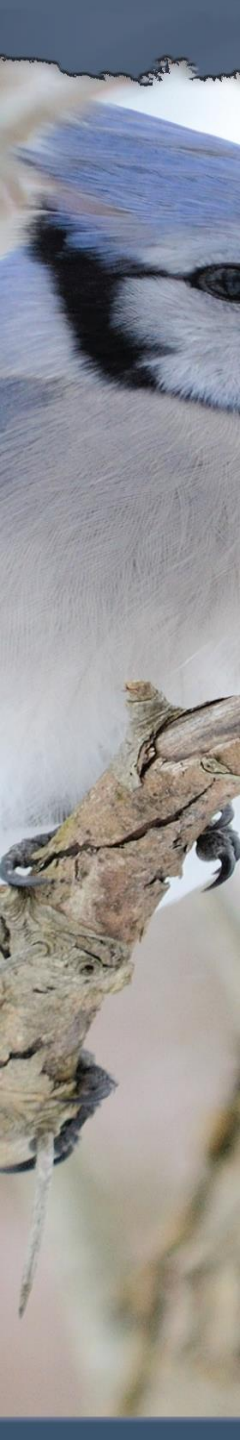
Seat

Seat number

Telephone number

Row

Row number

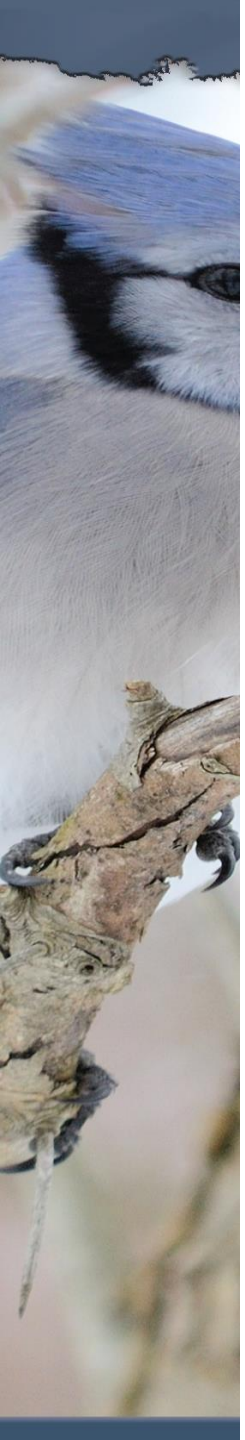


Using CRC cards

- First described by Kent Beck and Ward Cunningham.
- Each index card records:
  - A *class* name.
  - The class's *responsibilities*.
  - The class's *collaborators*.

A CRC card

Class name	**Collaborators**
Responsibilities	



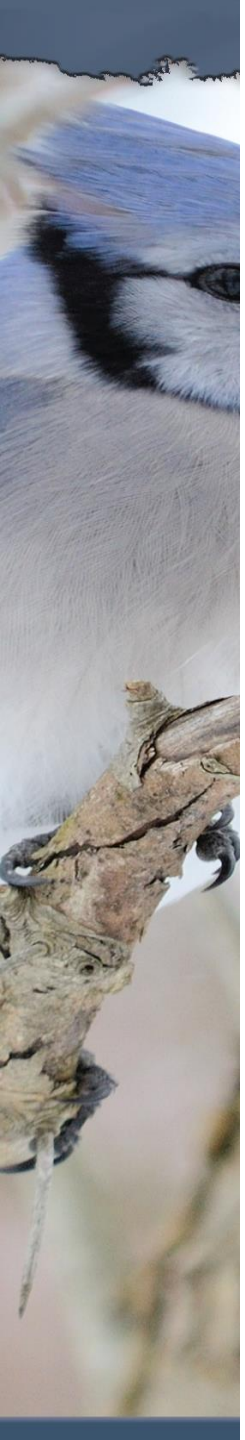
Scenarios

- An activity that the system has to carry out or support.
  - Sometimes known as *use cases*.
- Used to discover and record object interactions (collaborations).
- Can be performed as a group activity.

A partial example

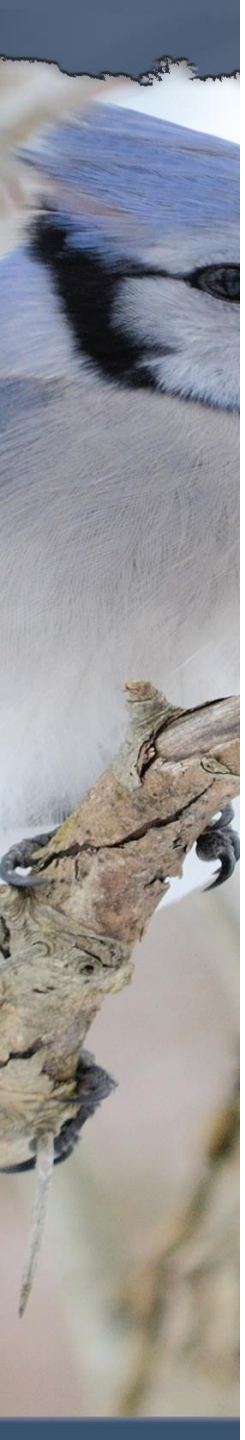
CinemaBookingSystem	Collaborators

Can find shows by title and day.
Stores collection of shows.
Retrieves and displays show details.
...
	`Show`
	
	`Collection`



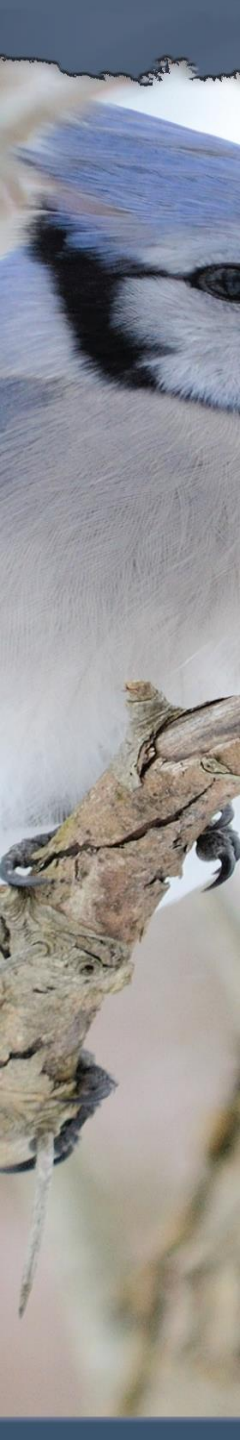
Scenarios as analysis

- Scenarios serve to check the problem description is clear and complete.
- Sufficient time should be taken over the analysis.
- The analysis will lead into design.
  - Spotting errors or omissions here will save considerable wasted effort later.



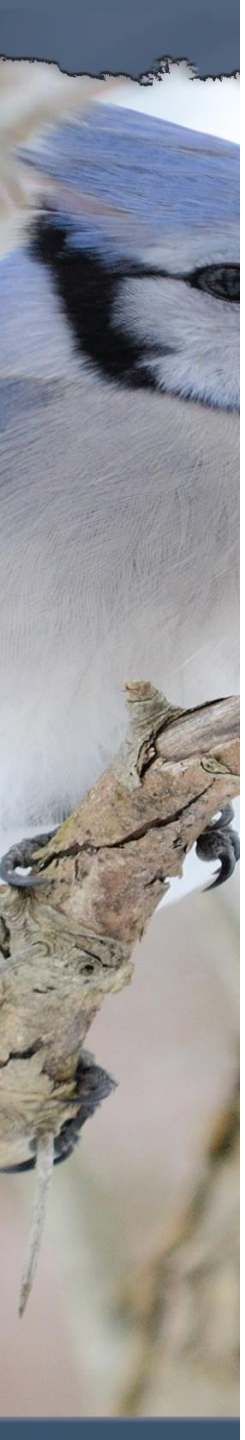
Class design

- Scenario analysis helps to clarify application structure.
  - Each card maps to a class.
  - Collaborations reveal class cooperation/object interaction.
- Responsibilities reveal public methods.
  - And sometimes fields; e.g., “Stores collection ...”



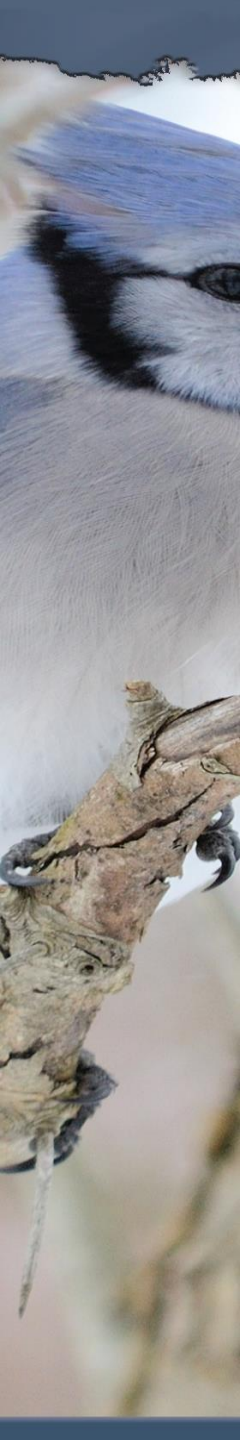
Designing class interfaces

- Replay the scenarios in terms of method calls, parameters and return values.
- Note down the resulting headers.
- Create outline classes with public-method stubs.
- Careful design is a key to successful implementation.



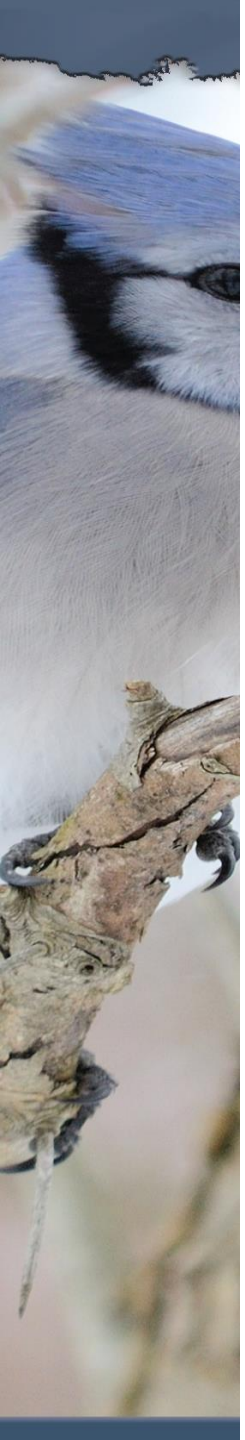
Documentation

- Write class comments.
- Write method comments.
- Describe the overall purpose of each.
- Documenting now ensures that:
  - The focus is on *what* rather than *how*.
  - That it doesn't get forgotten!



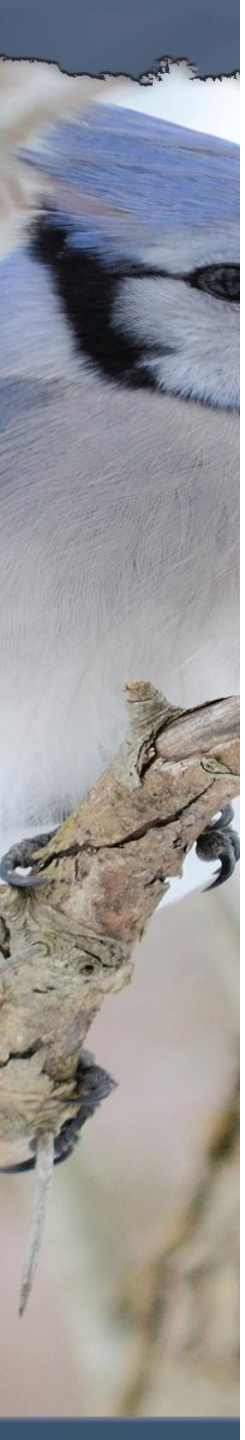
Cooperation

- Team-working is likely to be the norm not the exception.
- Documentation is essential for team working.
- Clean O-O design, with loosely-coupled components, also supports cooperation.



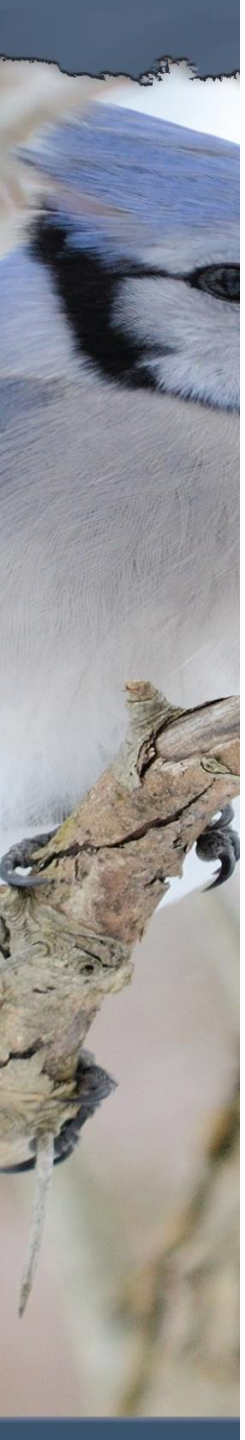
Prototyping

- Supports early investigation of a system.
  - Early problem identification.
- Incomplete components can be simulated.
  - E.g., always returning a fixed result.
  - Avoid random behavior which is difficult to reproduce.



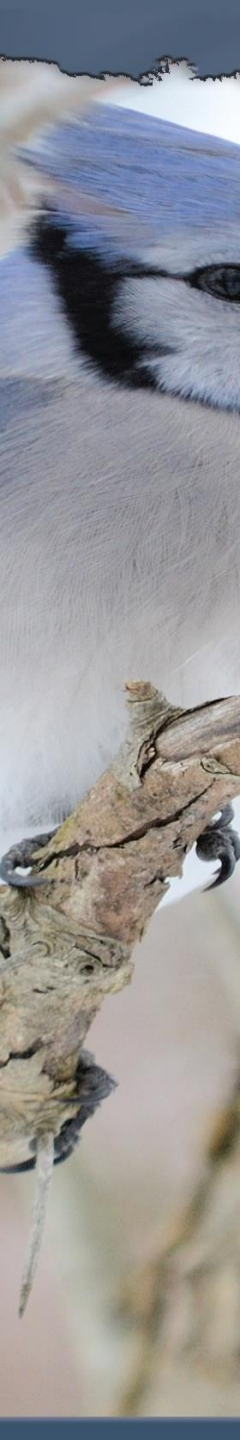
Software growth

- Waterfall model.
  - Analysis
  - Design
  - Implementation
  - Unit testing
  - Integration testing
  - Delivery
- No provision for iteration.



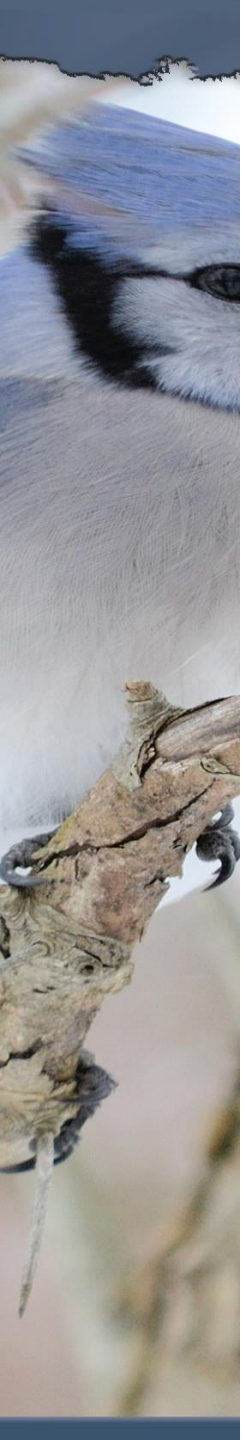
Iterative development

- Use early prototyping.
- Frequent client interaction.
- Iteration over:
  - Analysis
  - Design
  - Prototype
  - Client feedback
- A growth model is the most realistic.



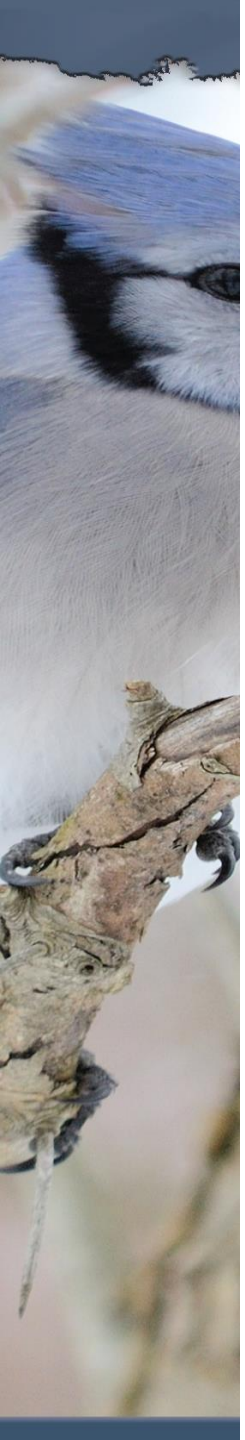
Using design patterns

- Inter-class relationships are important, and can be complex.
- Some relationships recur in different applications.
- Design patterns help clarify relationships, and promote reuse.



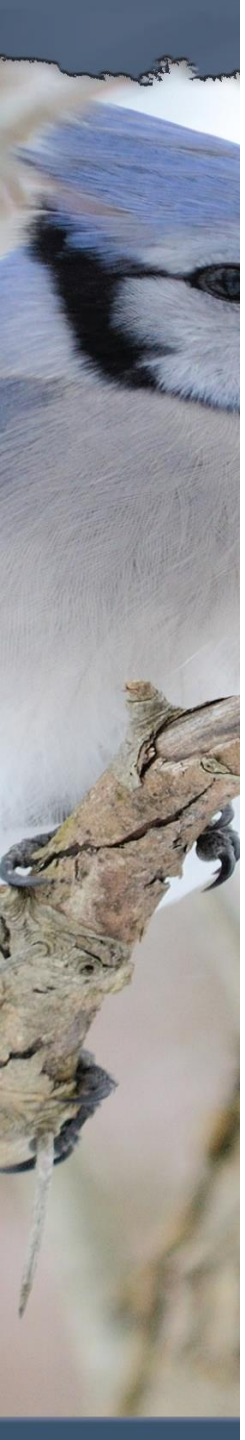
Pattern structure

- A pattern name.
- The problem addressed by it.
- How it provides a solution:
  - Structures, participants, collaborations.
- Its consequences.
  - Results, trade-offs.



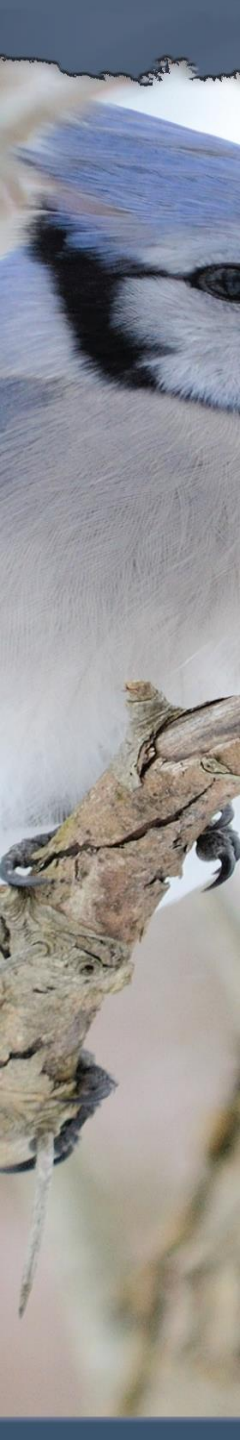
Decorator

- Augments the functionality of an object.
- Decorator object wraps another object.
  - The Decorator has a similar interface.
  - Calls are relayed to the wrapped object ...
  - ... but the Decorator can interpolate additional actions.
- Example: **`java.io.BufferedReader`**
  - Wraps and augments an unbuffered **`Reader`** object.



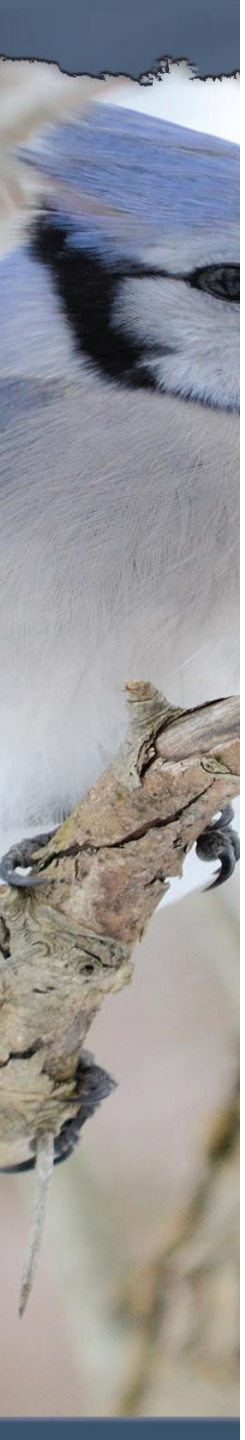
Singleton

- Ensures only a single instance of a class exists.
  - All clients use the same object.
- Constructor is private to prevent external instantiation.
- Single instance obtained via a static **`getInstance`** method.
- Example: **`Canvas`** in *shapes* project.



Factory method

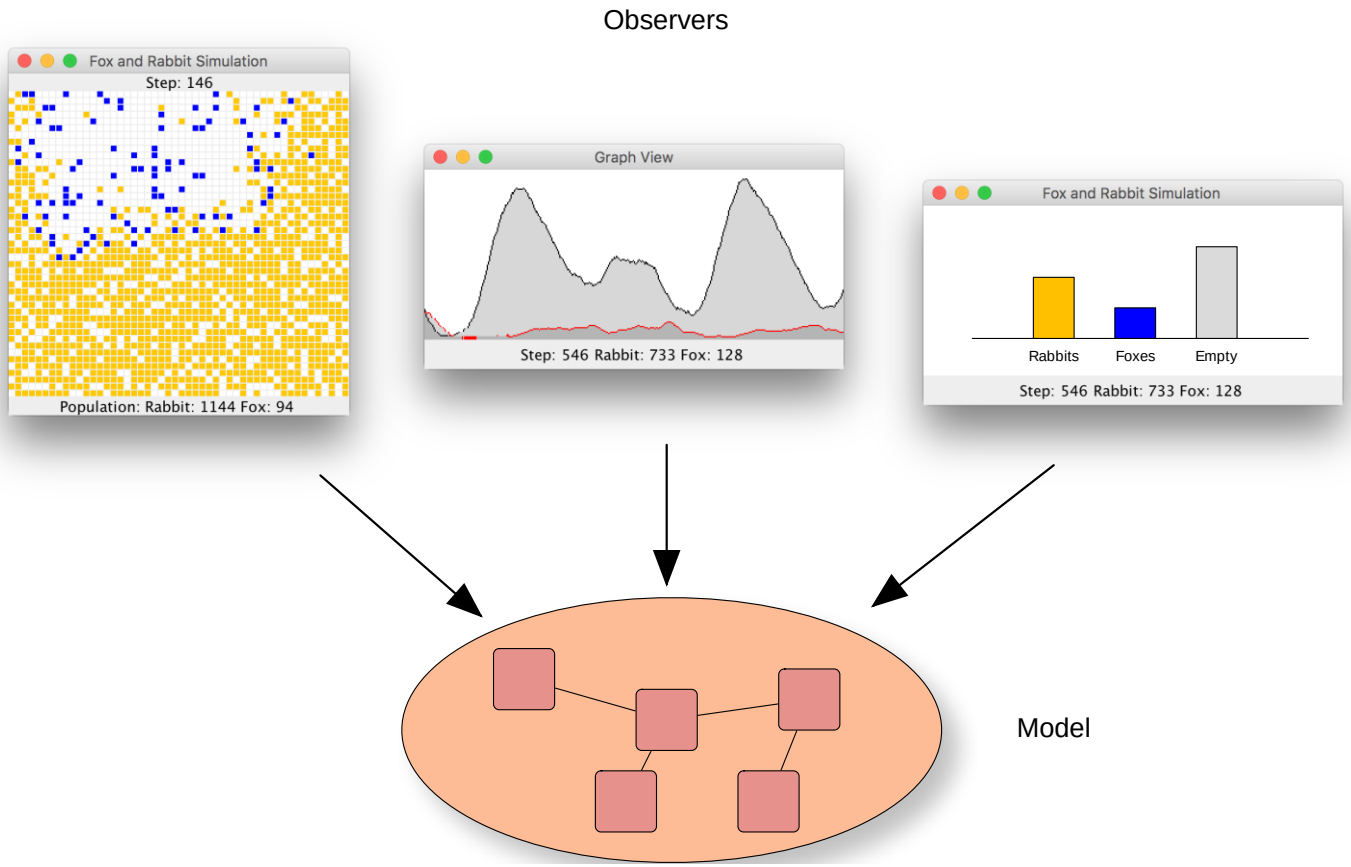
- A creational pattern.
- Clients require an object of a particular interface type or superclass type.
- A factory method is free to return an implementing-class object or subclass object.
- Exact type returned depends on context.
- Example: `iterator` methods of the collection classes.

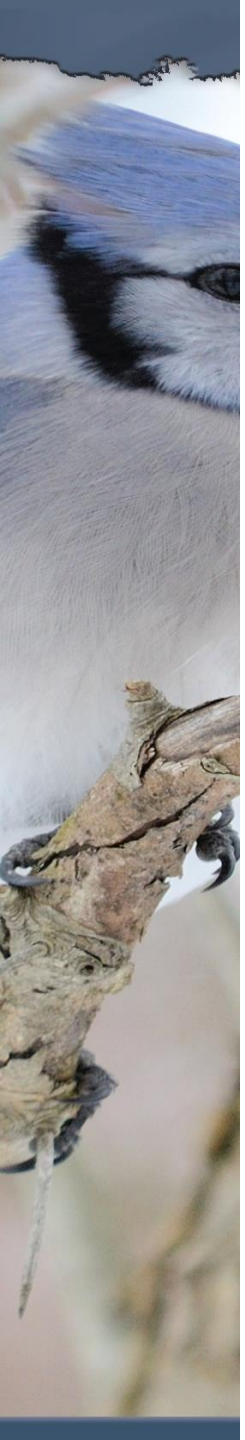


Observer

- Supports separation of internal model from a view of that model.
- Observer defines a one-to-many relationship between objects.
- The object-observed notifies all Observers of any state change.
- Example **`SimulatorView`** in the *foxes-and-rabbits project*.

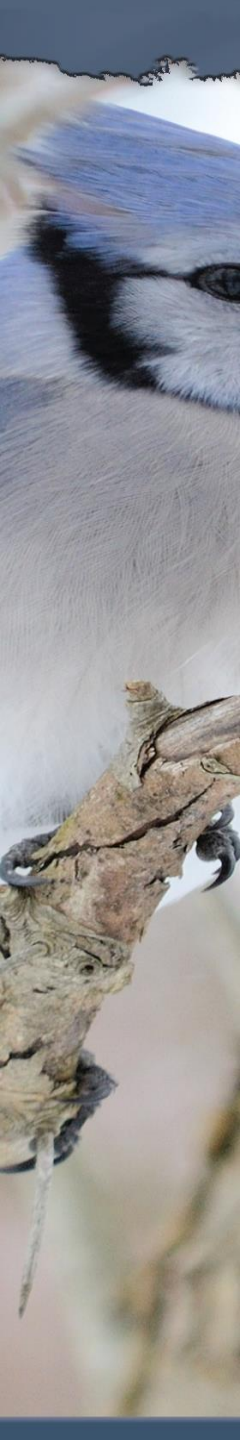
Observers





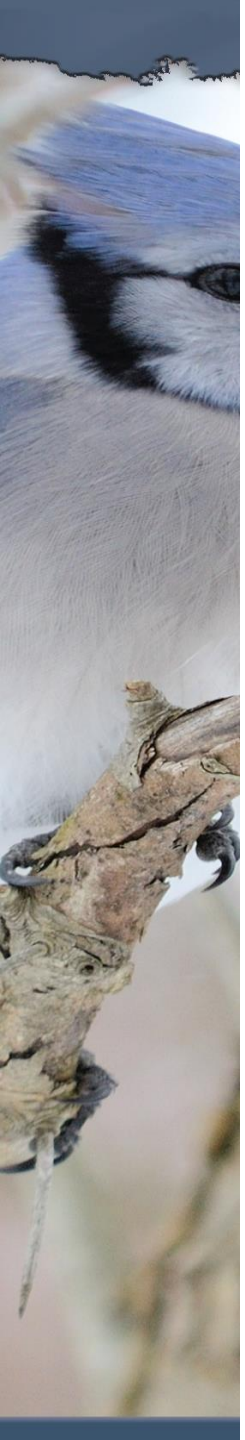
Review

- Class collaborations and object interactions must be identified.
  - CRC analysis supports this.
- An iterative approach to design, analysis and implementation can be beneficial.
  - Regard software systems as entities that will grow and evolve over time.



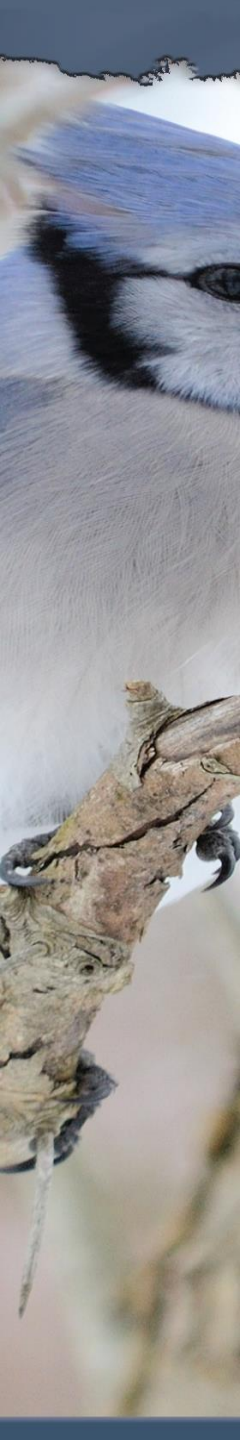
Review

- Work in a way that facilitates collaboration with others.
- Design flexible, extendible class structures.
  - Being aware of existing design patterns will help you to do this.
- Continue to learn from your own and others' experiences.



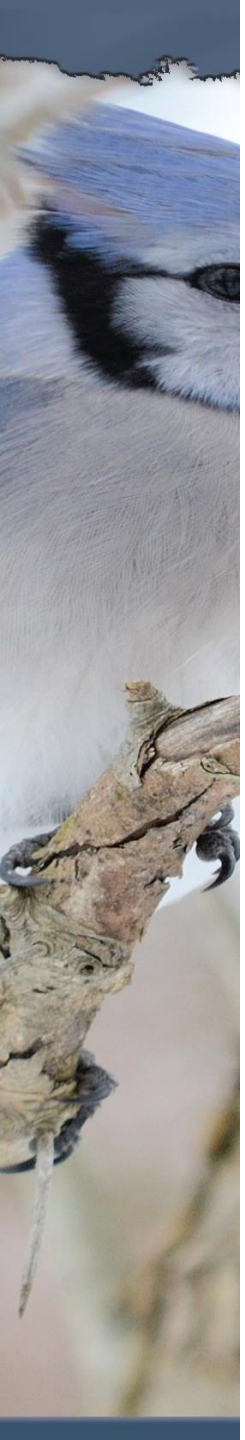
Dagens øving

- Fungerer mye bedre i gruppe enn individuelt.
- Bruk de gruppene dere allerede har, eller lag en gruppe i dag. Kanskje gruppen kan benyttes fram mot eksamen også?



Godbiter fra arbeidskrav 1

- Per demonstrerer…



Nå

- Kahoot☺
- Deretter øving på her Fjerdingen (sjekk TimEdit for rom)