From Domain Classes and Use Cases to Design Classes using CRC Cards

CRC Cards Technique ("Responsibility-Driven Design")

- Informal, non-detailed
- Used for group brain-storming
- End result is a first cut at **classes** for an object-oriented model,
- Not intended to provide a **complete** design

Use-Case Input

- A good **starting point** for CRC analysis is a clear statement of all of the use-cases.
- Use-cases drive the introduction of CRC cards.
- Use-cases, or their accompanying scenarios, can be used as a kind of script for the **role-playing method** of checking the CRC cards.
- The role-playing could be replaced with sequence diagrams.

CRC

- Stands for:
  - **C**lasses
  - **R**esponsibilities
  - **C**ollaborations

- (Not as in "CRC Handbook": "Chemical Rubber Company")

CRC

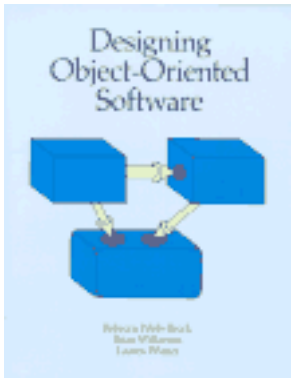
- Stands for:
  - **C**lasses (of objects)
  - **R**esponsibilities (of the objects in each class)
  - **C**ollaborations (with objects in other classes)
    - In UML, these will be examples of "associations"
- Remember that an application may have "singleton" classes (classes instantiated only once).

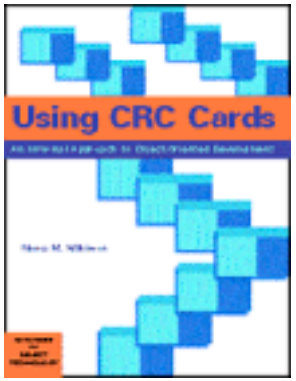
Origin of CRC

- Kent Beck and Ward Cunningham, formerly of Tektronix in Oregon
- Rebecca Wirfs-Brock popularized with "Responsibility-Driven Design" (RDD)

References

- Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener, *Designing Object-Oriented Software*, Prentice-Hall, 1990.
- David Bellin and Susan Simone, *The CRC Card Book*, Addison Wesley Longman, 1997.





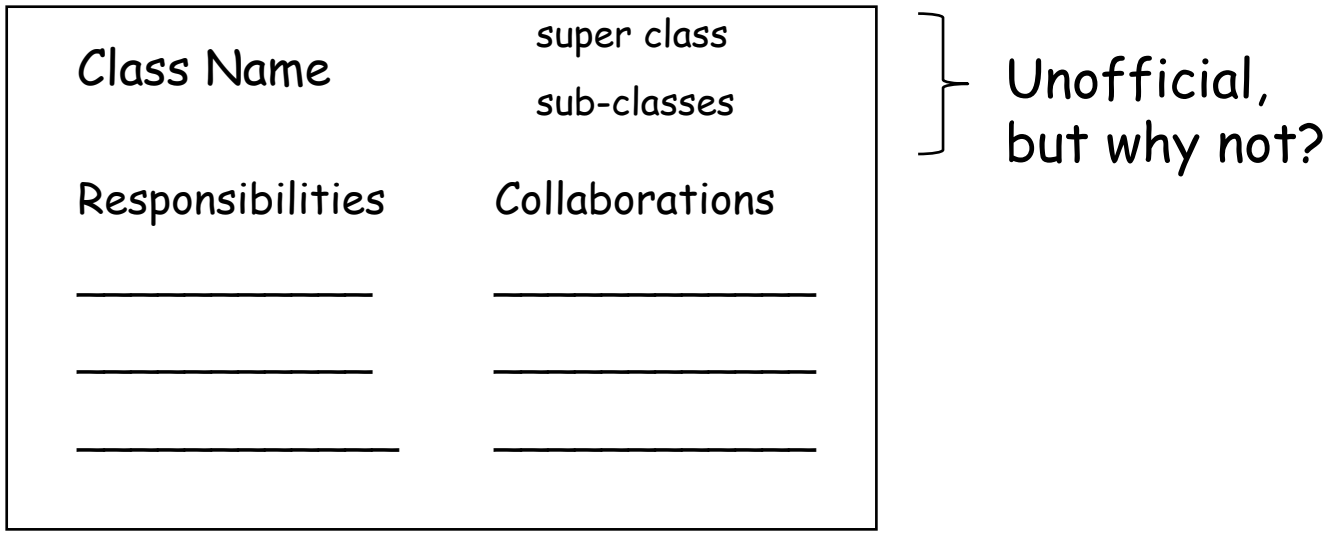
The Basic Idea

- Develop set of cards or cards images.
- Each card represents one class.
- A card contains:
  - The **name** of the class.
  - The **responsibilities** of the class.
  - **Collaborations**: other classes with which this class inter-operates, in conjunction with the attendant responsibility.

CRC cards represent a ***static*** view of the system's classes

- Domain class diagrams similarly static.
- Must eventually be augmented by dynamic description, e.g. sequence diagrams or other.
- Informal dynamic description can be acted out with "role-playing", similar to the creation of scenarios for use cases.

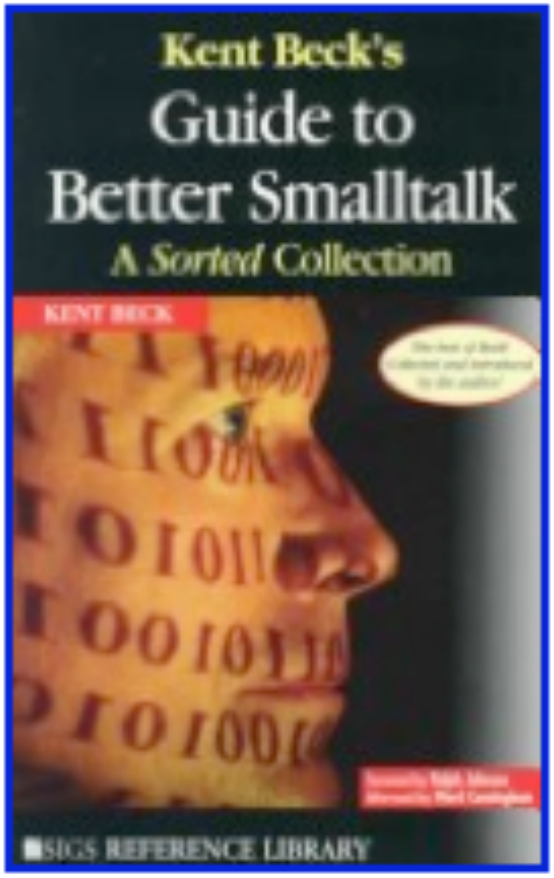
Image of CRC cards



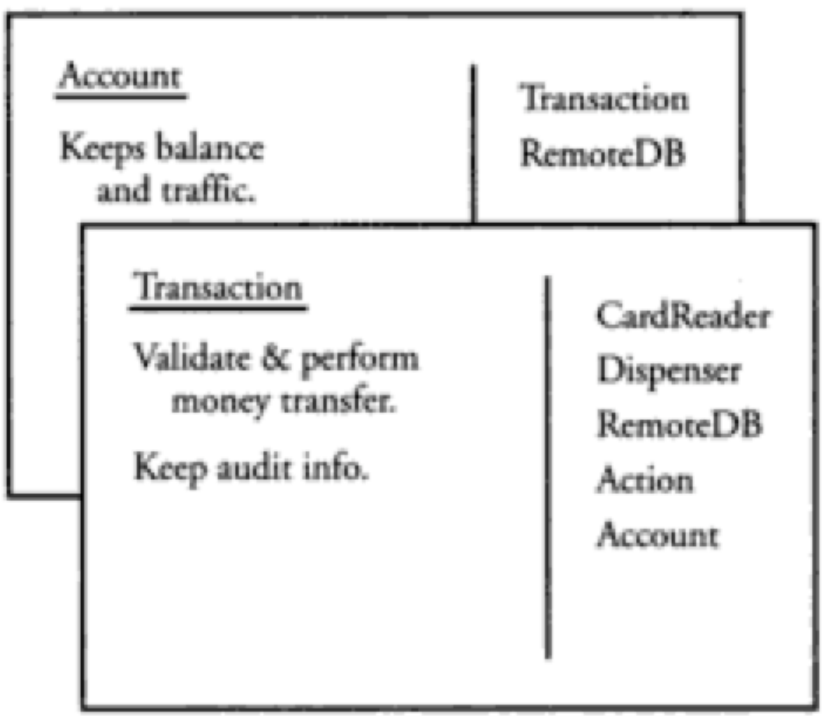
Use index cards, or perhaps single PowerPoint slides.

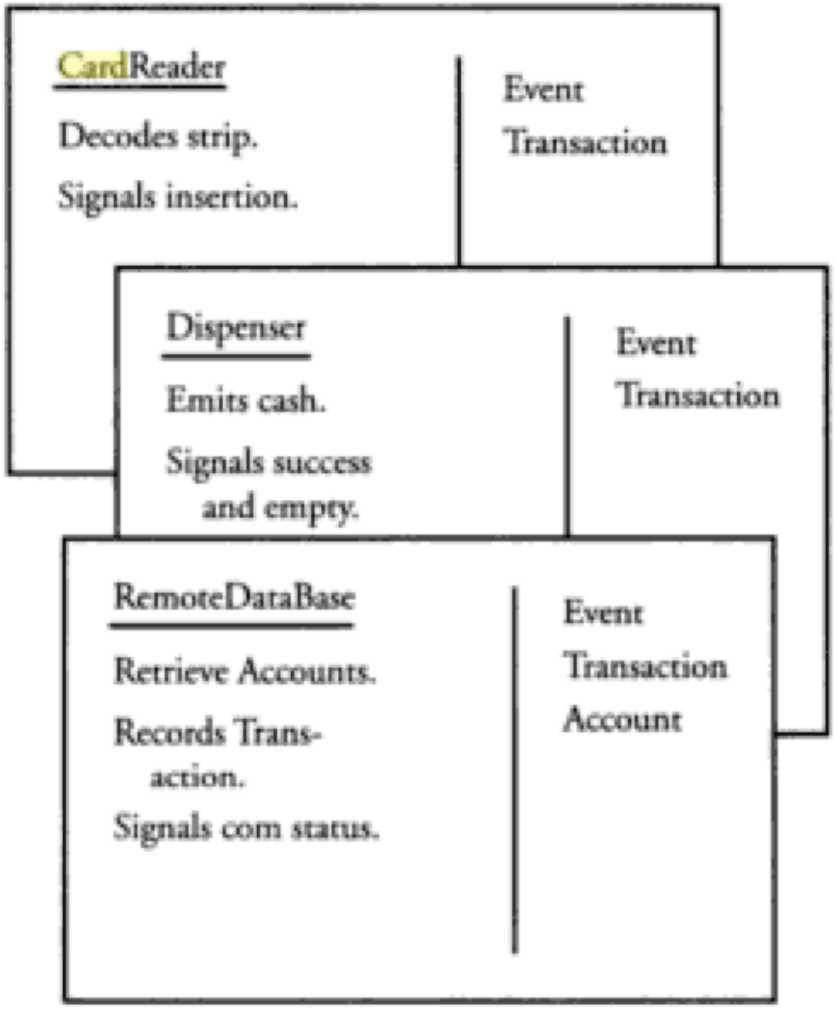
Limiting the size of a card is an attempt at preventing the class from becoming too complex.

Examples from Kent Beck



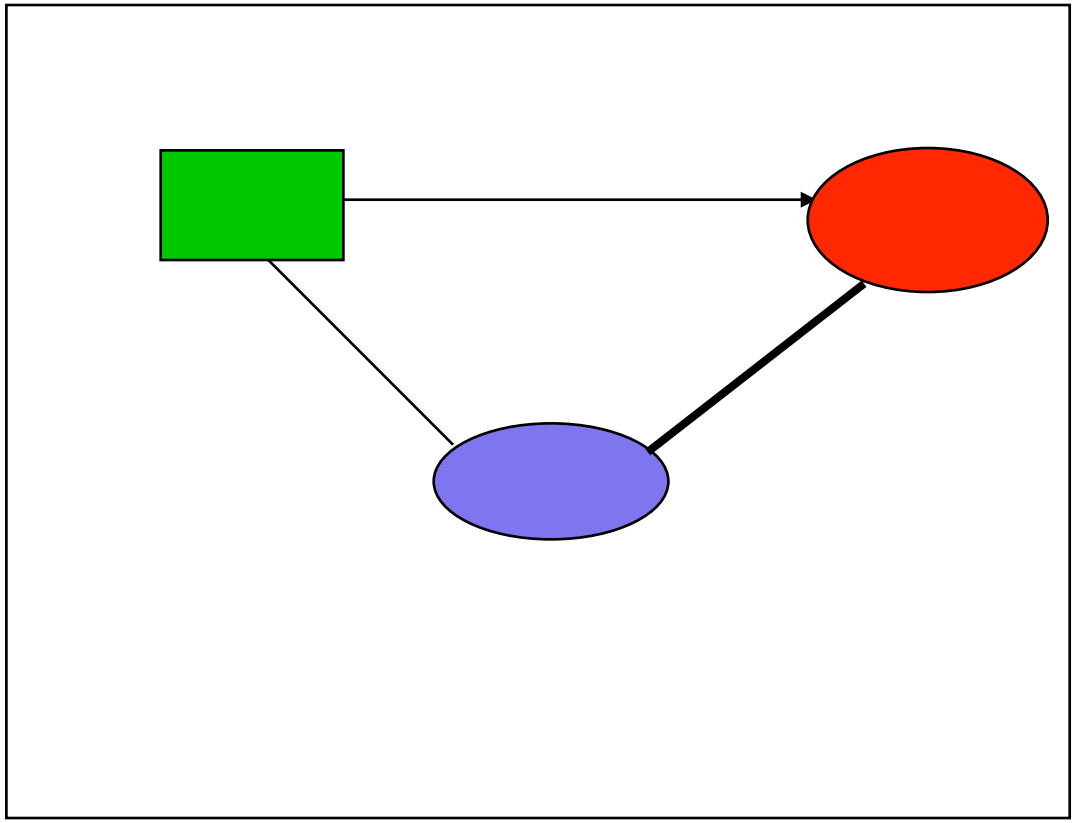
Examples from Kent Beck





Sample Application:
A graph-drawing program

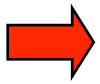
Possible screen image



Typical Application Use-Cases:

- **Draw shape**
- **Move shape**
- **Resize shape**
- **Connect shapes**
- **Erase shape**
- **Erase connector**

Example of CRC card
for a graph-drawing program (1)

Class 

Shape

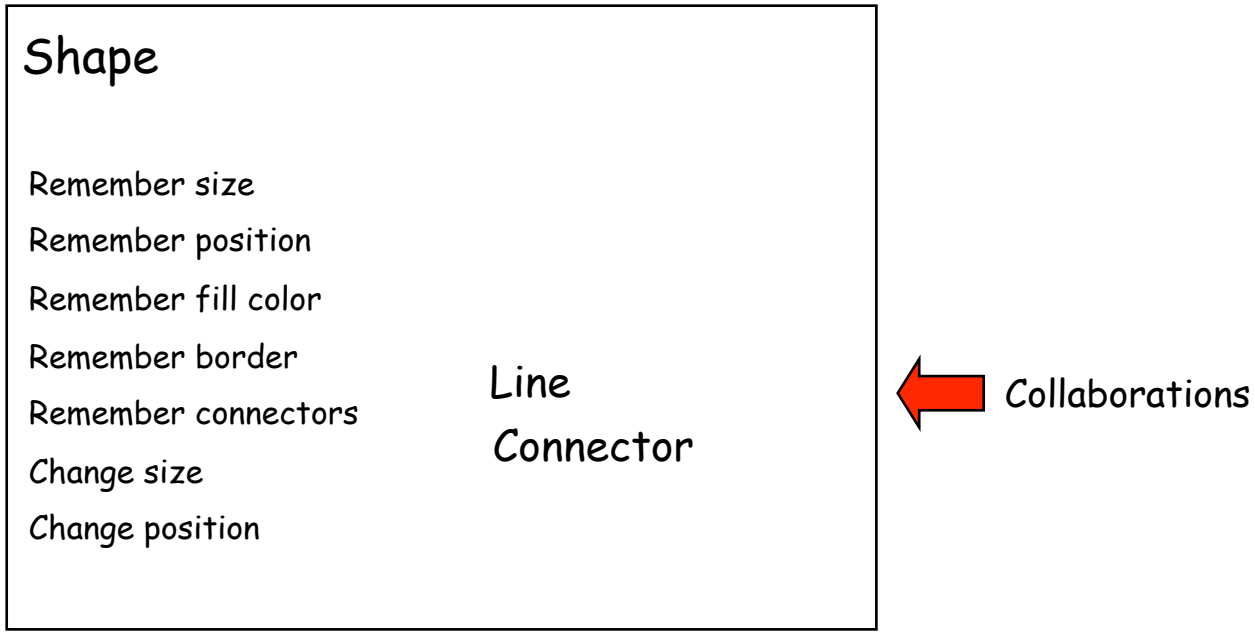
Example of CRC card
for a graph-drawing program (2)

Responsibilities

Shape

Remember size
Remember position
Remember fill color
Remember border
Remember connectors
Change size
Change position

Example of CRC card
for a graph-drawing program (3)

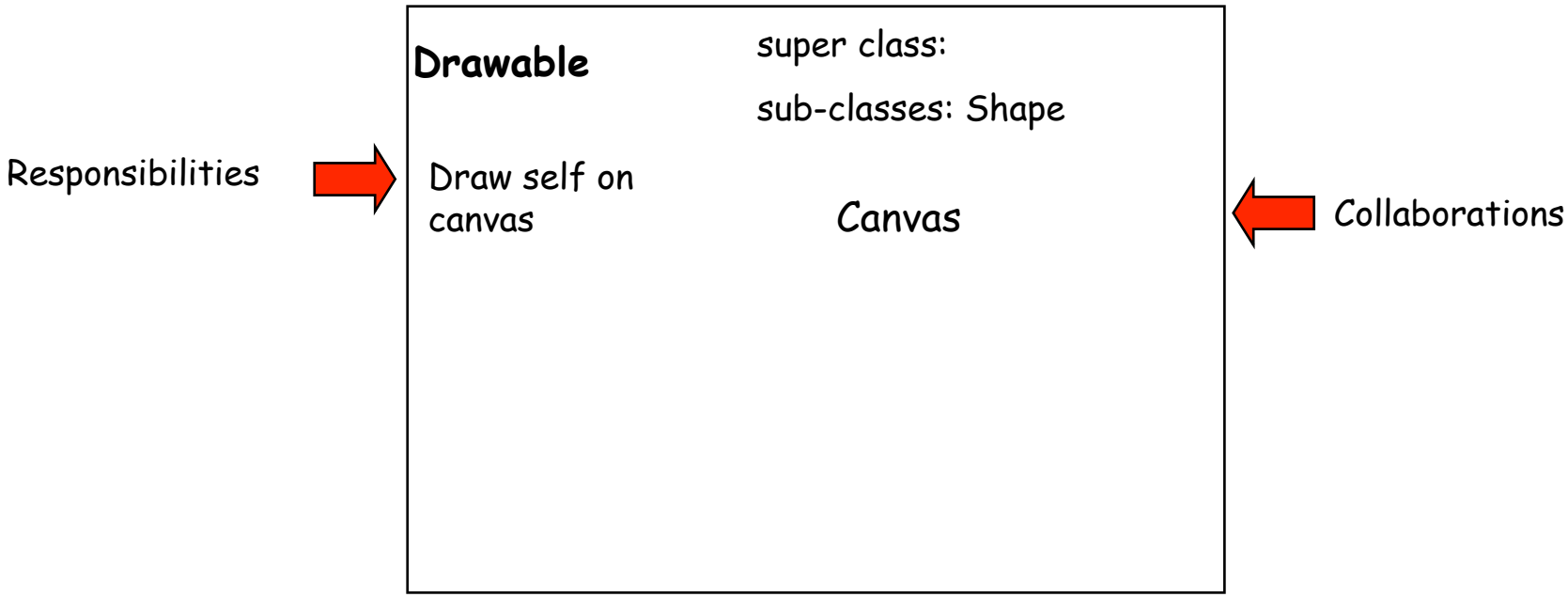


Example of CRC card
for a graph-drawing program (4)

Shape	super class: Drawable sub-classes: Rect, Oval, Group
Remember size Remember position Remember fill color Remember border Remember connectors Change size Change position	Line Connector

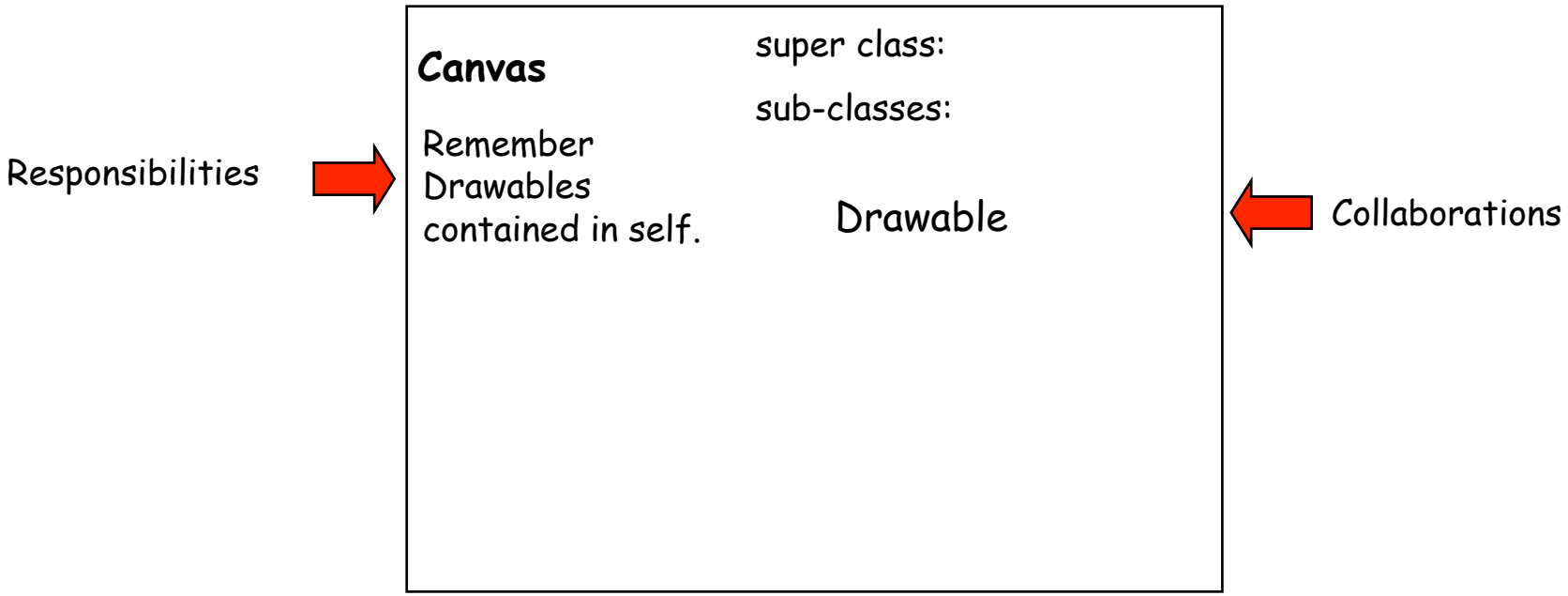
Super- and
Sub-classes

Example of CRC card
for a graph-drawing program (5)



Note: The Drawable doesn't necessarily need to *remember* a Canvas, since the Canvas could be passed as an argument to the *draw* method.

Example of CRC card
for a graph-drawing program (6)



Note:

- Responsibilities are usually for ***members*** (objects) of the class rather than the class itself, although

- Class-wide responsibility is possible (corresponding to **static** method)

Attribute Value vs. Object

- An object of a class typically has one or more ***attributes***.
- Attributes have ***values*** that specify or describe the object.
- A value might or might not deserve the distinction of being an object itself; It depends on what we intend to do with the attribute.
- A would-be attribute that is object-valued is actually a ***collaboration*** (***association*** in UML).

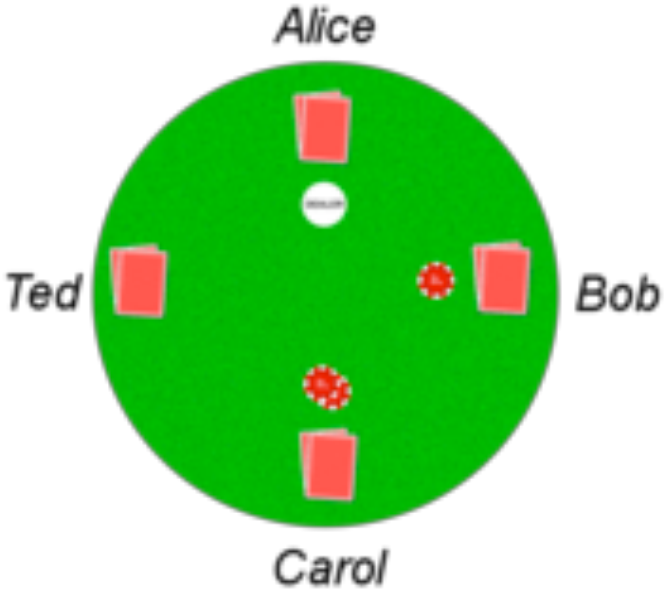
Immutable Objects

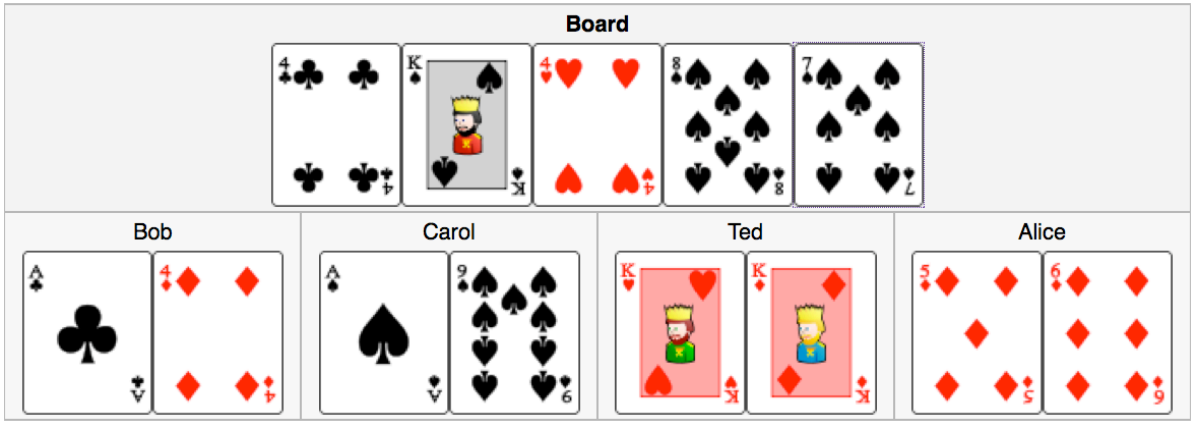
- Some objects only "know", but don't "do" anything.
- They can't be changed once created, and therefore are called *immutable*.
- Can immutable objects have collaborations?

Exercise

- We want to design a simulator for a card game, say Texas Hold'em Poker.
- What would you suggest for:
  - The classes
  - Their responsibilities
  - Their collaborations
- Start with the classes, then add the others and revise as needed.

Texas Hold'em





Kent Beck: Advice for Good CRC Classes

- Have a consistent set of names, often taken from a physical metaphor. For instance, if you were designing an object-oriented graphics model you might choose the metaphor of painting on canvas. The names of your classes would be taken from common words painters use. You wouldn't call one kind of paint "Paint" and another "ColorValue."

- Have terse, strongly worded responsibilities. As your object designing matures, your objects will add behavior to your designs and not just act as holders of data for other objects to manipulate.

Kent Beck: Advice for Good CRC Classes

- Have a small, sensible set of collaborators. "Indispensable" objects that talk to everyone in the world are a sign of global, procedural thinking. Break up such objects, with by distributing their responsibilities to existing objects or creating several new objects to take their place.

- Fit comfortably on an index card. If a card is crowded, find a more concise way of stating the object's responsibilities. Failing that, the object is doing too much. Find or create other objects to take on some of its tasks.

Kent Beck: Worse CRC Classes

- Don't have any responsibilities. It's fine to create an object with a name but no responsibilities, trusting that its behavior will become clear as the design progresses. Once a design is finished, however, discard any object that doesn't carry its share of the computational load.

- Have the word "Manager" in their name. Words like "Manager," "Object," or "Thing" add nothing to the meaning of the object's name. If you can't just eliminate the offending "noise" word, find a better word from the metaphor domain or substitute a word that tells what the object is doing. For example, use ProcessScheduler rather than ProcessManager or Figure rather than DrawingObject.

Kent Beck: Worse CRC Classes

- Use the words "has," "holds," or "uses" in their responsibilities. These words suggest representation but not behavior. Replace the responsibility with one that explains why the other object is held. For example, in a Bitmap object you might list as one of its responsibilities "Holds width and height." You can rewrite it more actively by saying "Shapes bits into a rectangle."

- Have large, highly connected clumps. If an object takes care of a limited part of the computation it can't need to talk to lots of other objects. Good designs have small clusters of objects with limited connections between them. Shuffle responsibilities to achieve this kind of interconnection pattern.

CRC Team Structure

- Usually ≤ 6 person team is recommended
- The team can include clients as well as developers (even though we are partly in the design phase)
  - 1-2 domain experts
  - 1-2 analysts
  - experienced object-oriented designer
  - leader

Once the CRC cards are constructed …

- Team can engage in **role-playing** to verify that use-case **scenarios** make sense for chosen CRC.
- Each person can role-play one or more class cards.
- If something doesn't work, change the class accordingly.
- Revision of use-cases might also be indicated.

Use-Case to Class Traceability Matrix Example (from the graph-drawing example)

Class: Responsibility

Use Case	Drawing: remember components	Shape: draw	Shape: remember position	Shape: remember size	Shape: remember connectors	Connector: draw	Connector: remember start	Connector: remember end
Draw shape	x	x	x	x				
Move shape		x	x		x			
Erase shape	x				x		x	x
Resize shape		x	x	x	x			
Connect shapes	x				x	x	x	x
Erase connector	x				x			

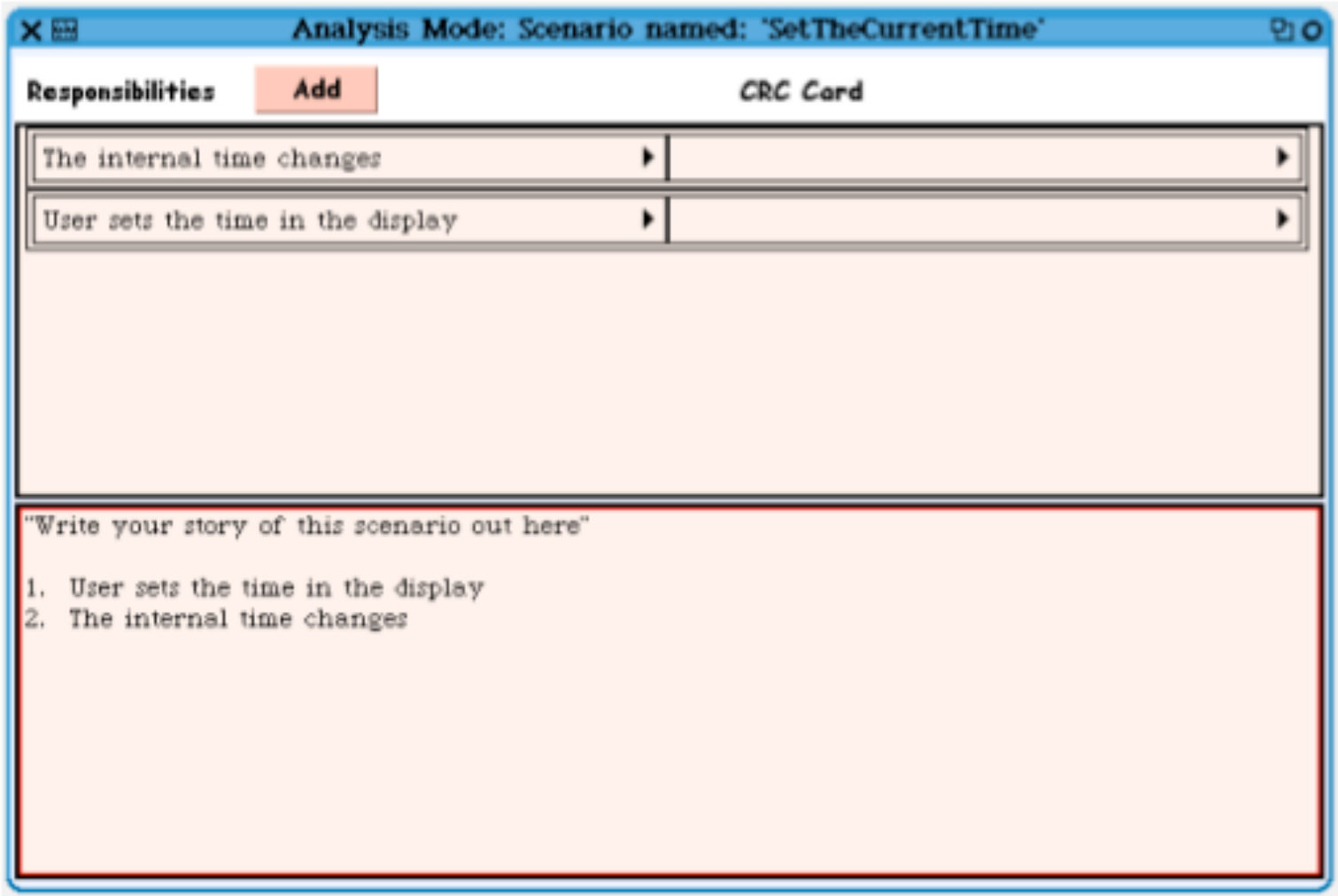
Examples of CRC-based Process and Tools

- ECoDE [Georgia Tech]
- Ectropic Collaborative Design Environment
- Kathleen Arnold Gray and Mark Guzdial and Spencer Rugaber
- "Ectropic" = opposite of "Entropic"
- International Conference on Software Engineering (ICSE-99) Workshop on Software Change and Evolution.
  Los Angeles, May 17, 1999

Extending CRC Cards into a Complete Design Process

"We teach CRC cards as one part of a design process that begins with brainstorming class names and developing usage scenarios, and continues through use of UML diagrams [2]. Students are less excited about the rest of the process. Their scenarios tend to be ill-defined and too brief. Students tell us frankly on anonymous surveys that they typically complete the UML class diagrams after they finish the code."

CRC Card with Scenario



Adding Methods