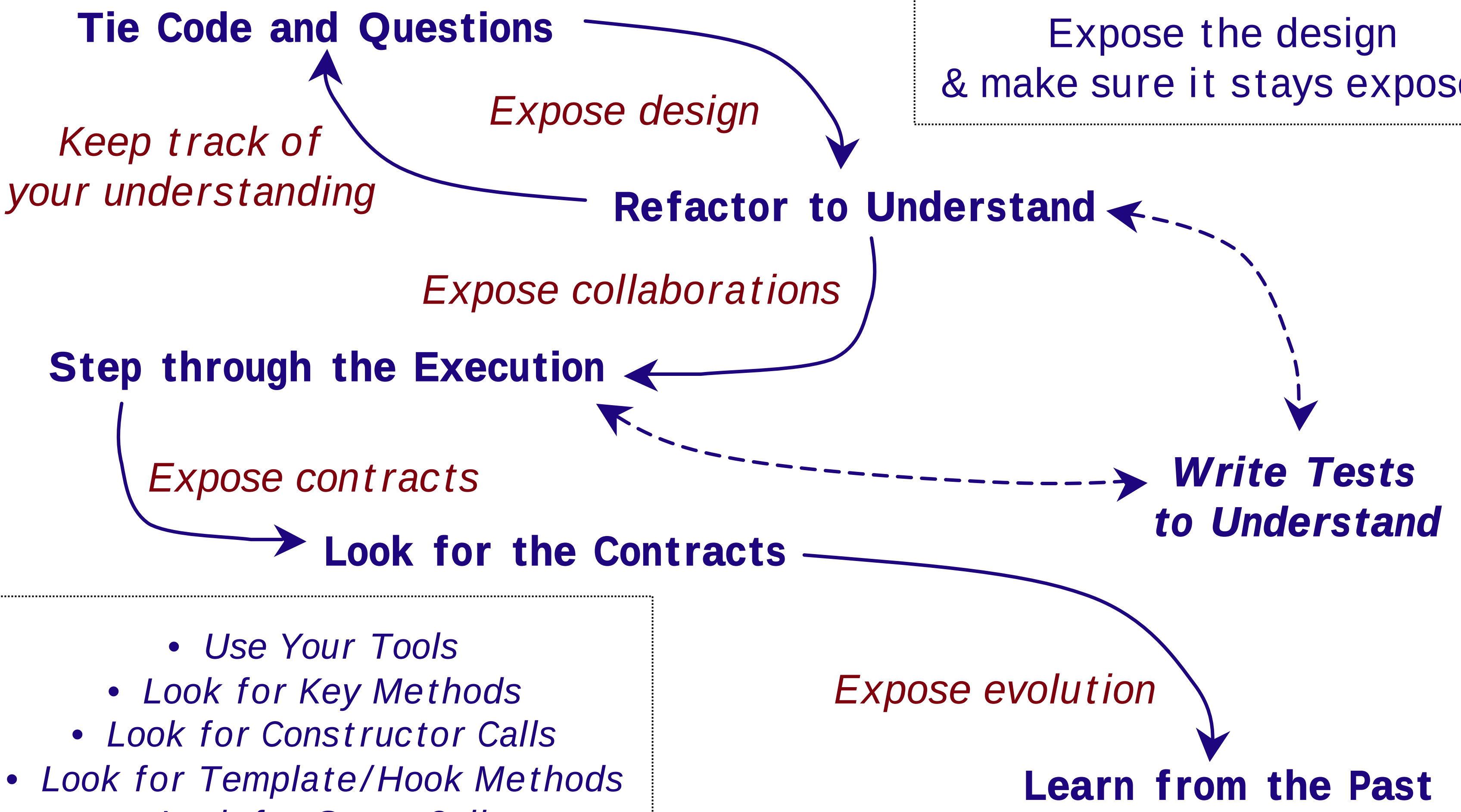
Detailed Model Capture

- Details matter
  - *Pay attention to the details!*
- Design remains implicit
  - *Record design rationale when you discover it!*
- Design evolves
  - *Important issues are reflected in changes to the code!*
- Code only exposes static structure
  - *Study dynamic behavior to extract detailed design*

Detailed Model Capture



Expose the design
& make sure it stays exposed

- *Use Your Tools*
- *Look for Key Methods*
- *Look for Constructor Calls*
- *Look for Template/Hook Methods*
- *Look for Super Calls*

Tie Code and Questions

Problem: How do you keep track of your understanding?

Solution: Annotate the code

- List questions, hypotheses, tasks and observations.
- Identify yourself!
- Annotate as *comments*, or as *methods*

Refactor to Understand

Problem: How do you decipher cryptic code?

Solution: Refactor it till it makes sense

- Goal (for now) is to *understand*, not to reengineer
- Work with a *copy* of the code
- Refactoring requires an adequate test base
  - If this is missing, *Write Tests to Understand*
- ...and tool support
  - automatic refactorings
- Hints:
  - Rename attributes to convey roles
  - Rename methods and classes to reveal intent
  - Remove duplicated code
  - Replace condition branches by methods
  - Define method bodies with same level of abstraction
- Needs tool support!

Look for the Contracts

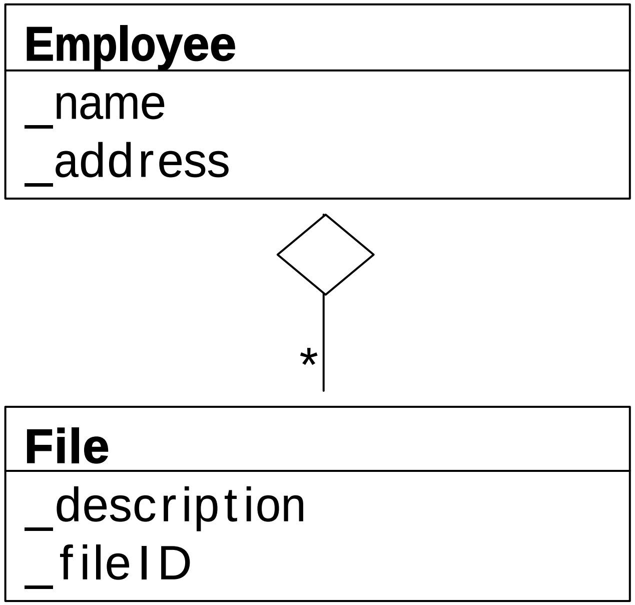
Problem: Which contracts does a class support?

Solution: Look for common programming idioms, i.e. look for "customs" of using the interface of that class

- Look for “*key methods*”
  - Intention-revealing names
  - Key parameter types
  - Recurring parameter types represent temporary associations
- Look for *constructor* calls
- Look for *Template/Hook* methods
- Look for *super* calls
- *Use your tools!*

Constructor Calls: Stored Result

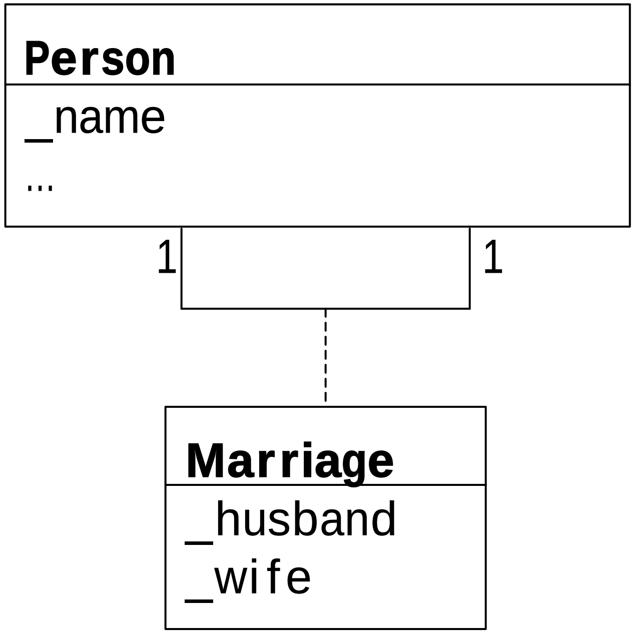
```
public class Employee {
   private String _name = "";
   private String _address = "";
   public File[ ] files = { };
...
public class File {
   private String _description = "";
   private String _fileID = "";

...
public void createFile (int position, String description, String identification)
{
      files [position] = new File (description, identification);
}
```



- Identify part-whole relationships (refining associations)
  - storing result of constructor in attribute ⇒ part-whole relation

Constructor Calls: "self" Argument

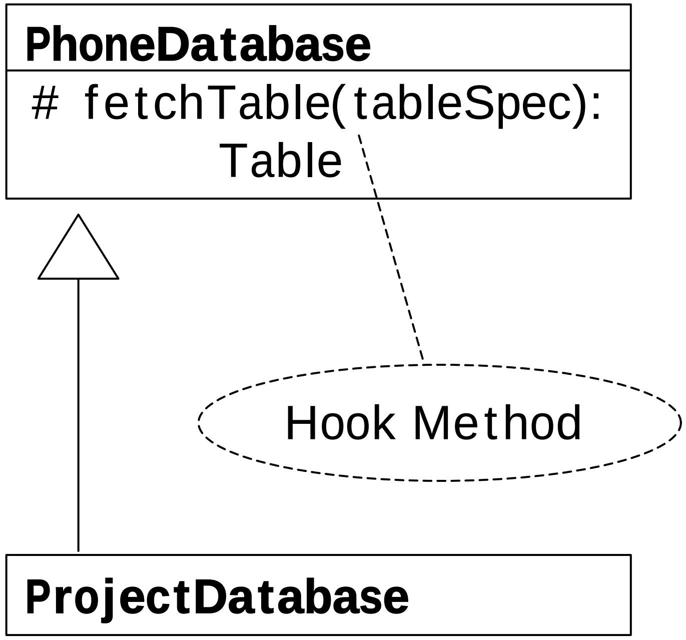
```
public class Person {
   private String _name = "";
...
public class Marriage {
   private Person  _husband, _wife;
   public Marriage (Person husband,
              Person wife) {
              _husband = husband;
              _wife = wife;}
...
```



```
Person::public Marriage marryWife (Person wife) {
  return new Marriage (this, wife);
}
```

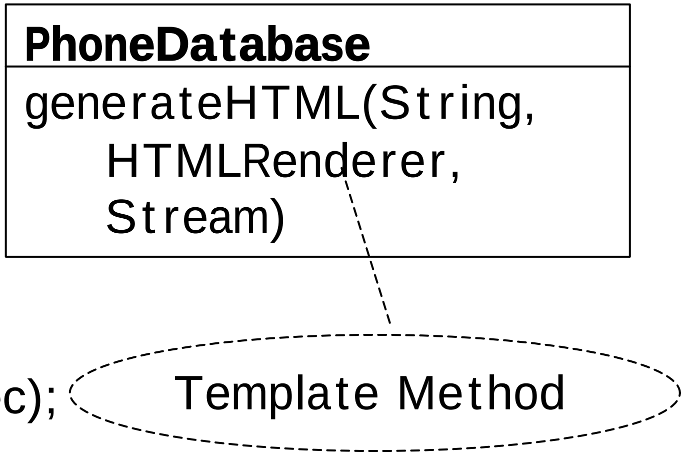
Hook Methods

```
public class PhoneDatabase {
    ...
    protected Table fetchTable (String tableSpec) {
    //tableSpec is a filename; parse it as
    //a tab-separated table representation
    ...};

public class ProjectDatabase
        extends PhoneDataBase {
    ...
    protected Table fetchTable (String tableSpec) {
    //tableSpec is a name of an SQLTable;
    //return the result of SELECT * as a table
    ...};
```



Template / Hook Methods

```
public class PhoneDatabase {
     ...
     public void generateHTML
               (String tableSpec,
               HTMLRenderer aRenderer,
               Stream outStream) {
          Table table = this.fetchTable (tableSpec);
          aRenderer.render (table, outStream);}
...};
```



```
public class HTMLRenderer {
     ...
     public void render (Table table, Stream outStream) {
     //write the contents of table on the given outStream
     //using appropriate HTML tags
...}
```

Learn from the Past

Problem: How did the system get the way it is?

Solution: Compare versions to discover where code was removed

- *Removed* functionality is a sign of design evolution
- Use or develop appropriate *tools*
- Look for signs of:
  - *Unstable design* — repeated growth and refactoring
  - *Mature design* — growth, refactoring and stability

Step Through the Execution

Problem: How do you uncover the run-time architecture?

Solution: *Execute scenarios of known use cases and step through the code with a debugger*

- Difficulties
  - OO source code exposes a *class hierarchy*, not the run-time *object collaborations*
  - Collaborations are spread throughout the code
  - Polymorphism may hide which classes are instantiated
- Focussed use of a debugger can expose collaborations

Conclusion

- *Setting Direction* + *First Contact*
  ⇒ First Project Plan

- *Initial Understanding* + *Detailed Model Capture*
  - Plan the work … and Work the plan
  - Frequent and Short Iterations

- Issues
  - scale
  - speed vs. accuracy
  - politics