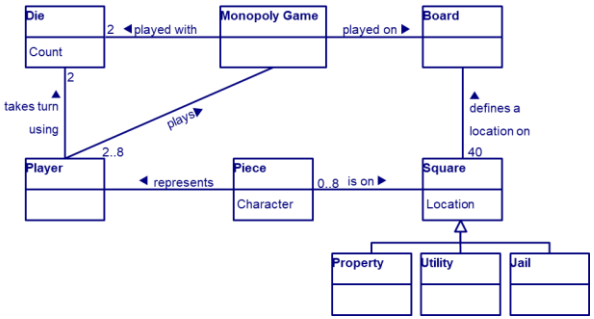
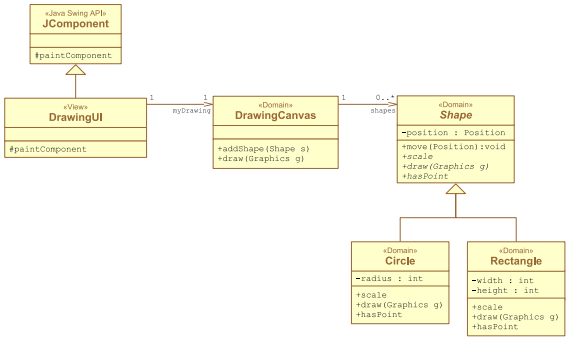
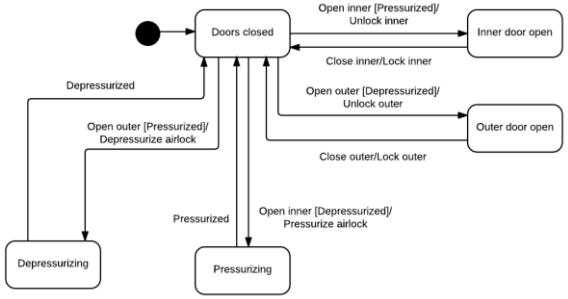
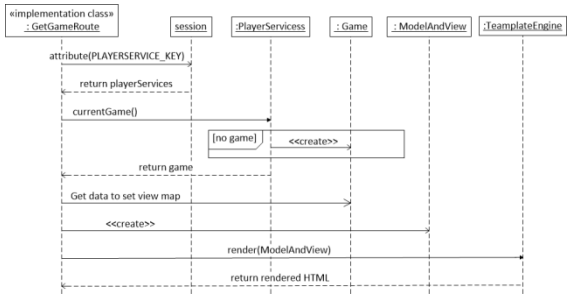
Design and Code Communication









SWEN-261 Introduction to Software Engineering

Department of Software Engineering
Rochester Institute of Technology


```
/**
  * Get the {@linkplain GuessGame game} for the current user.
  * The user is identified by a {@linkplain Session browser session}.
  *
  * @param session
  *   The HTTP {@link Session}, must not be null
  *
  * @return
  *   An existing or new {@link GuessGame}
  *
  * @throws NullPointerException
  *   when the session parameter is null
  */
 public GuessGame get(final Session session)
```

Software Engineering
Rochester Institute of Technology

Your communication about a project is not just in the form of presentations and meetings.

- The systems that you will develop are complex and have both static and dynamic design characteristics.
- To describe those characteristics you will use several UML models.
  - ***Domain, class, statechart, sequence***
- Those who must use your implementation need a more productive description that studying lines of code.
- Those who must maintain your implementation must be able to quickly understand the code.

The domain model describes the product owner's understanding of the application's scope.

- Domain model
  - ***Describes the context in which the application will operate.***
  - ***Helps developers share the product owner's understanding of this context.***
  - ***Describes the product owner's world view of the domain entities and relationships between them.***
- The domain model will help developers create a structure for the implementation to the extent that is possible.

The class model defines the static structure of your implementation.

- It captures many constructs embodied in your implementation
  - ***Class attributes and methods with visibilities***
  - ***Relationships between classes with multiplicities***
  - ***Navigation between classes***
  - ***Structure via inheritance/interface***
  - ***Architectural tiers***
- The domain model inspires the first-cut for the implementation class structure.
  - ***Try to have the software structure match the product owner's domain structure, i.e. domain entities become implementation classes***

You also must describe the application's dynamic characteristics to fully describe its operation.

- The dynamic behavior is often state-based and succinctly described with a statechart.
  - ***Exchanges between a client and web application***
  - ***User interface operation***
  - ***Communication protocols***
  - ***Individual classes with state-based characteristics***
- An application's execution of a feature/operation involves multiple classes across architectural tiers.
  - ***The sequence diagram indicates which classes and methods are involved in an execution scenario.***
  - ***Formulate a user story solution with one or more sequence diagrams created before starting the implementation.***

How your code "reads" is critically important for the humans who will read it.

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Refactoring: Improving the Design of Existing Code
Martin Fowler, et. al (1999)

Code is read by humans as much as by machines.

- Code must be readable and understandable by all team members.
- Clear code communication includes:
  - ***A shared code style***
  - ***Use of good, meaningful names***
  - ***Component APIs are clearly documented***
  - ***Algorithms are clarified using in-line comments***
  - ***Indication of incomplete or broken code***

A shared code style is good etiquette.

- No code style is inherently better than any other one.
- A code style includes:
  - ***Spaces vs tabs***
  - ***Where to put curly-braces***
  - ***Naming conventions***
    - `CamelCase` for class names
    - `UPPER_CASE` for constants
    - `lowerCamelCase` for attribute and method names
  - ***And so on***
- Every team should choose a style and stick to it.
  - ***IDEs provide support for defining a code style***
  - ***If your team cannot choose one then we recommend using Google Java style (see resources)***

Make names reflect what they mean and do.

- Dos:
  - ***Use names that reflect the purpose***
  - ***Use class names from analysis and domain model***
  - ***Use method names that are verbs in your analysis***
  - ***Use method names that describe what it does not how it does it***
- Don'ts:
  - ***Don't abbreviate; spell it out***
    - `pricePerUnit` is better than `pPU` or worse just `p`
  - ***Don't use the same local variable for two purposes; create a new variable with an appropriate name***
  - ***Don't use "not" in a name***
    - `isValid` is better than `isNotValid`.

Document your component's API.

- In Java the `/**  ...  */` syntax is used to denote a documentation for the thing it precedes.
- For example:

```
/**
 * A single "guessing game".
 *
 * @author <a href='mailto:joecool@rit.edu'>Joe Cool</a>
 */
public class GuessGame
```

- At a minimum you should document all public members.
  - ***Also good to document all methods including private methods***
  - ***Document attributes with complex data structures***

A method's javadoc must explain how to use the operation.

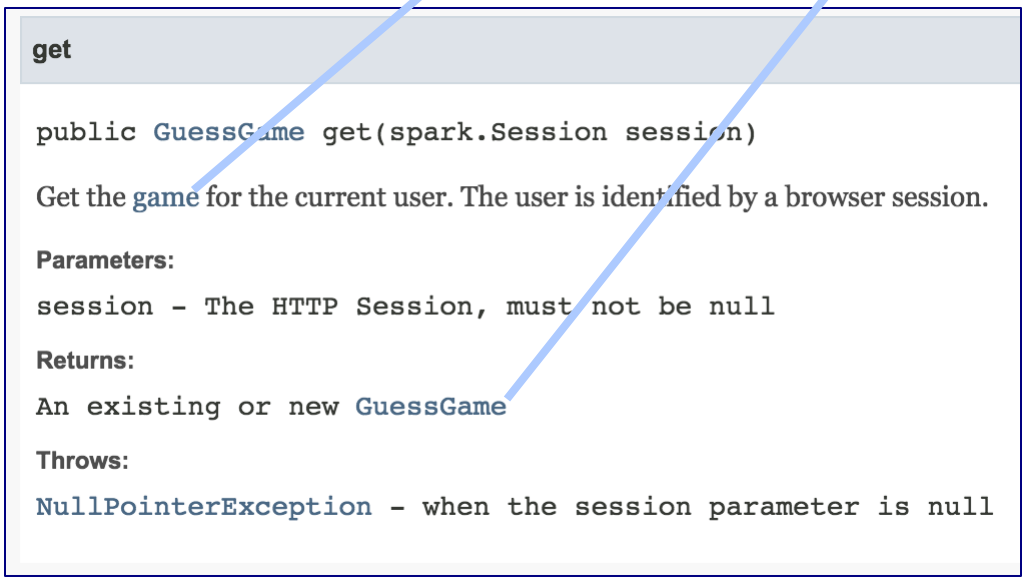
- Every method must have an opening statement that expresses what it does.
  - ***Keep this statement concise***
  - ***Additional statements can be added for clarification***
- Document the method signature
  - ***Use `@return` to describe what is returned***
  - ***Use `@param` to describe each parameter***
  - ***Use `@throws` to describe every exception explicitly thrown by the method***
- Link to other classes
  - ***Use `@link` to link to classes***
  - ***Use `@linkplain` in opening statement***

Example method javadocs.

```
/**
  * Get the {@linkplain GuessGame game} for the current user.
  * The user is identified by a {@linkplain Session browser session}.
  *
  * @param session
  *   The HTTP {@link Session}, must not be null
  *
  * @return
  *   An existing or new {@link GuessGame}
  *
  * @throws NullPointerException
  *   when the session parameter is null
  */
 public GuessGame get(final Session session)
```

Use @linkplain in the opening statement.

Use @link in all other clauses.



get

`public GuessGame get(spark.Session session)`

Get the game for the current user. The user is identified by a browser session.

Parameters:

`session - The HTTP Session, must not be null`

Returns:

`An existing or new GuessGame`

Throws:

`NullPointerException - when the session parameter is null`

Use in-line comments to communicate algorithms and intention.

- Use in-line comments to describe an algorithm
  - ***Dos:***
    - Use pseudo-code steps
    - Explain complex data structures
  - ***Don'ts:***
    - Don't repeat the code in English

```
count++; // increment the count
```

- Use comments to express issues and intentions
  - ***A TODO comment hints at a future feature***
  - ***A FIX (or FIXME) comment points to a known bug that is low priority***