Lecture 14

Design Patterns

What is a Design Pattern?

- A (Problem, Solution) pair.

- A technique to repeat designer success.
- Borrowed from Civil and Electrical Engineering domains.

How Patterns are used?

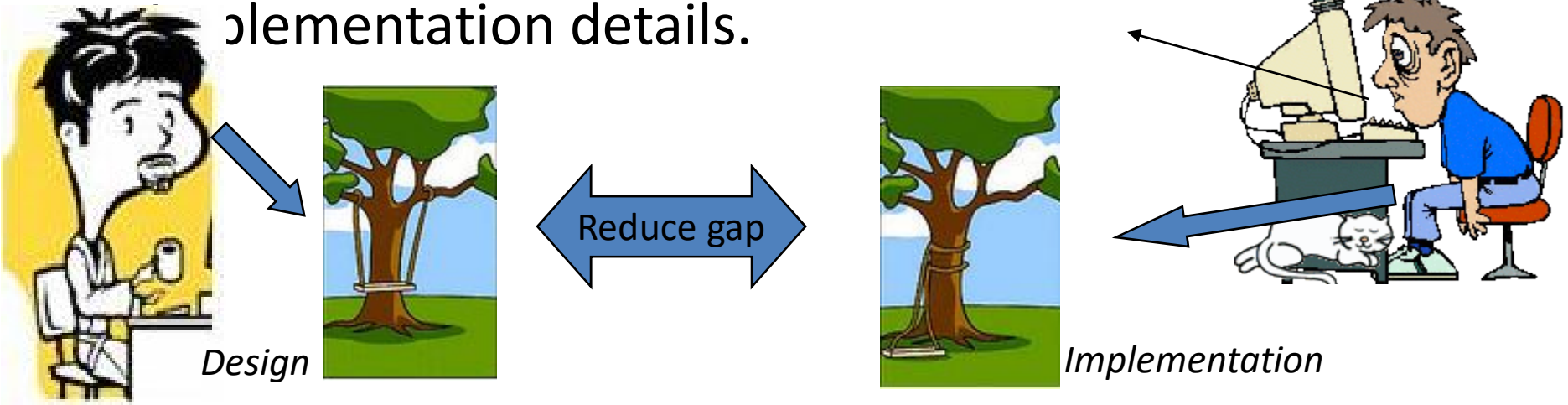
Designer

– Design Problem.

– Solution.

– Implementation details.

Programmer



Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design patterns: elements of reusable object-oriented software*. 1995.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: *Pattern-oriented software architecture: a system of patterns*. 2002.

Design patterns you have already seen

- Encapsulation (Data Hiding)
- Subclassing (Inheritance)
- Iteration
- Exceptions

Encapsulation pattern

- **Problem:** Exposed fields are directly manipulated from outside, leading to undesirable dependences that prevent changing the implementation.

- **Solution:** Hide some components, permitting only stylized access to the object.

Subclassing pattern

- **Problem:** Similar abstractions have similar members (fields and methods). Repeating these is tedious, error-prone, and a maintenance headache.

- **Solution:** Inherit default members from a superclass; select the correct implementation via run-time dispatching.

Iteration pattern

- **Problem:** Clients that wish to access all members of a collection must perform a specialized traversal for each data structure.

- **Solution:** Implementations perform traversals. The results are communicated to clients via a standard interface.

Exception pattern

- **Problem:** Code is cluttered with error-handling code.

- **Solution:** Errors occurring in one part of the code should often be handled elsewhere. Use language structures for throwing and catching exceptions.

Derived Conclusion

- ~~Patterns are Programming language features~~.
  - Programming languages are moving towards Design.
  - Many patterns are being implemented in programming languages.

Pattern Categories

- **Creational Patterns** concern the process of object creation.
- **Structural Patterns** concern with integration and composition of classes and objects.
- **Behavioral Patterns** concern with class or object communication.

What is the addressing Quality Attribute?

- Modifiability, Exchangeability, Reusability, Extensibility, Maintainability.

What properties these patterns provide?

- More general code for better Reusability.
- Redundant code elimination for better Maintainability.

What is the Singleton Pattern?

- The Singleton pattern ensures that a class is only instantiated once and provides a global access point for this instance

Creational Pattern

- Abstracts the instantiation process
- Object Creational Pattern
  - Delegates the instantiation to another object

Why use the Singleton Pattern?

- It is important for some classes to only have one instance
- It is also important that this single instance is easily accessible
- Why not use a global object?
  - Global variables make objects accessible, but they don't prevent the instantiation of multiple objects.

Solution

- Make the class responsible for keeping track of its only instance
  - The class can ensure that no other instances are created
  - This provides a useful way to access the instance
  - This is the Singleton pattern

Examples of Singleton Patterns

- Print Spooler
- File Systems
- Window Managers
- Digital Filters
- Accounting Systems

Sample Class

```
class Singleton
{
// static variable single_instance of type Singleton
private static Singleton single_instance = null;

// variable of type String
public String s;
```

Sample Implementation

```
// private constructor restricted to this class itself
private Singleton()
{
    s = "Hello I am a string part of Singleton class";
}
// static method to create instance of Singleton class



} e;
};
```

```
public static Singleton getInstance()
{
    if (single_instance == null)
        single_instance = new Singleton();

    return single_instance;
}
```

Things to Notice

- getInstance() function ensures that only one instance is created and that it is initialized before use
- Singleton constructor is declared as private
  - Direct instantiation causes errors at compile time

Benefits of the Singleton Pattern

- Controlled access to sole instance
- Reduced name space
- Allows refinement of operations and representation
- Allows a variable number of instances
- More flexible than class operations

Use the Singleton Pattern when...

- There must be exactly one instance of a class
  - This instance must be accessible from a well-known access point
- The instance should be extensible by subclassing
  - Clients should be able to use a derived class without modifying their code