Object-Oriented Design

Object-Oriented Design

Object-Oriented Design

Chapter Goals

- ✓To learn about the software life cycle
- ✓To learn how to discover new classes and methods
- ✓To be able to identify association, aggregation, composition and dependency relationships between classes
- ✓To master the use of UML class diagrams to describe class relationships
- ✓To learn how to use object-oriented design to build complex programs

The Software Life Cycle

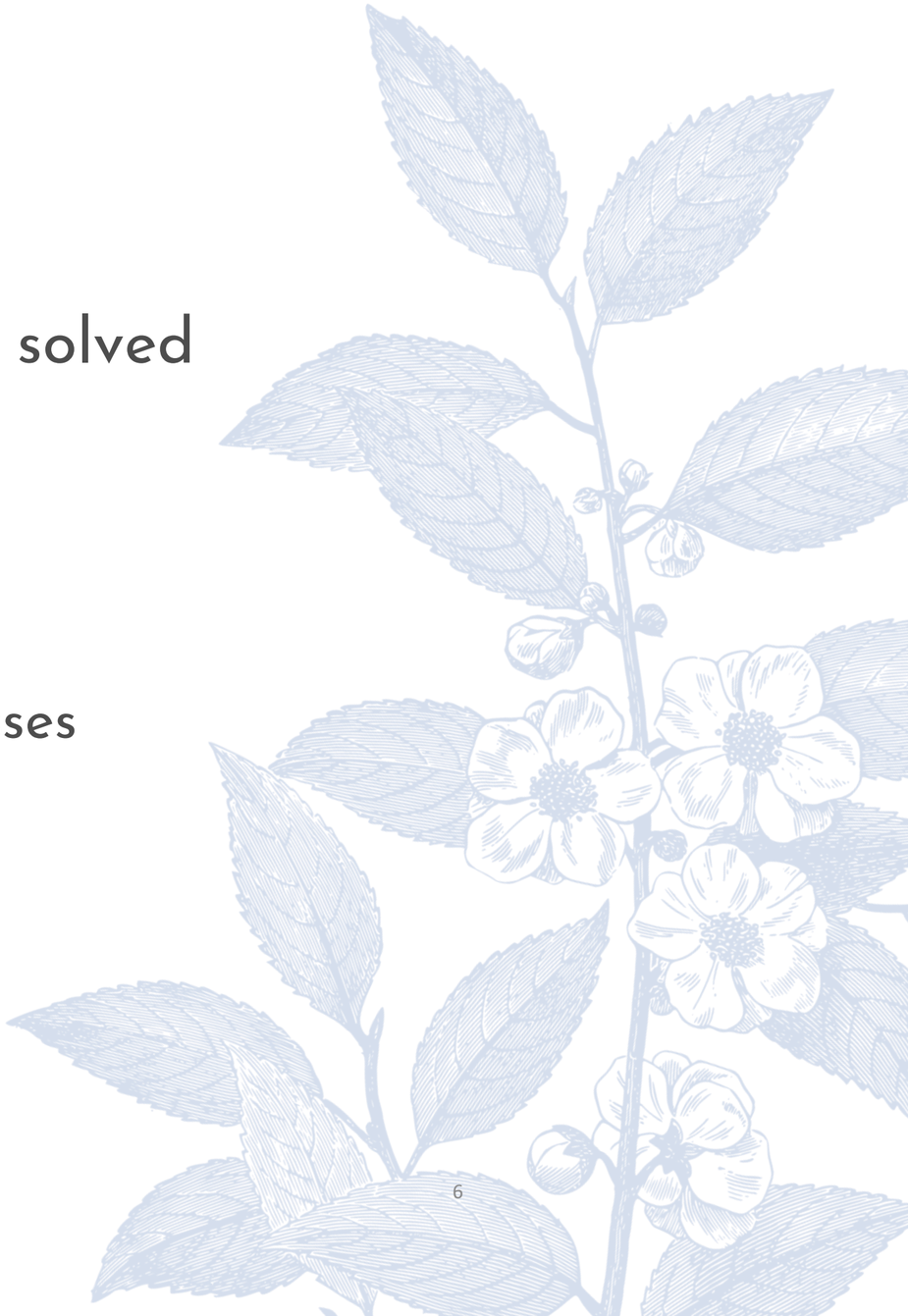
- ✓Encompasses all activities from initial analysis until obsolescence
- ✓Formal process for software development
  - Describes phases of the development process
  - Gives guidelines for how to carry out the phases
- ✓Development process
  - Analysis
  - Design
  - Implementation
  - Testing
  - Deployment

Analysis

- ✓Decide what the project is supposed to do
- ✓Do not think about how the program will accomplish tasks
- ✓Output: Requirements document
  - Describes what program will do once completed
  - User manual: Tells how user will operate program
  - Performance criteria

Design

- ✓Plan how to implement the system
- ✓Discover structures that underlie problem to be solved
- ✓Decide what classes and methods you need
- ✓Output:
  - Description of classes and methods
  - Diagrams showing the relationships among the classes



Implementation

- ✓Write and compile the code
- ✓Code implements classes and methods discovered in the design phase
- ✓Program Run: Completed program

Testing

- ✓Run tests to verify the program works correctly
- ✓Program Run: A report of the tests and their results

Deployment

- ✓Users install program
- ✓Users use program for its intended purpose

The Waterfall Model

- ✓Sequential process of analysis, design, implementation, testing, and deployment
- ✓When rigidly applied, waterfall model did not work

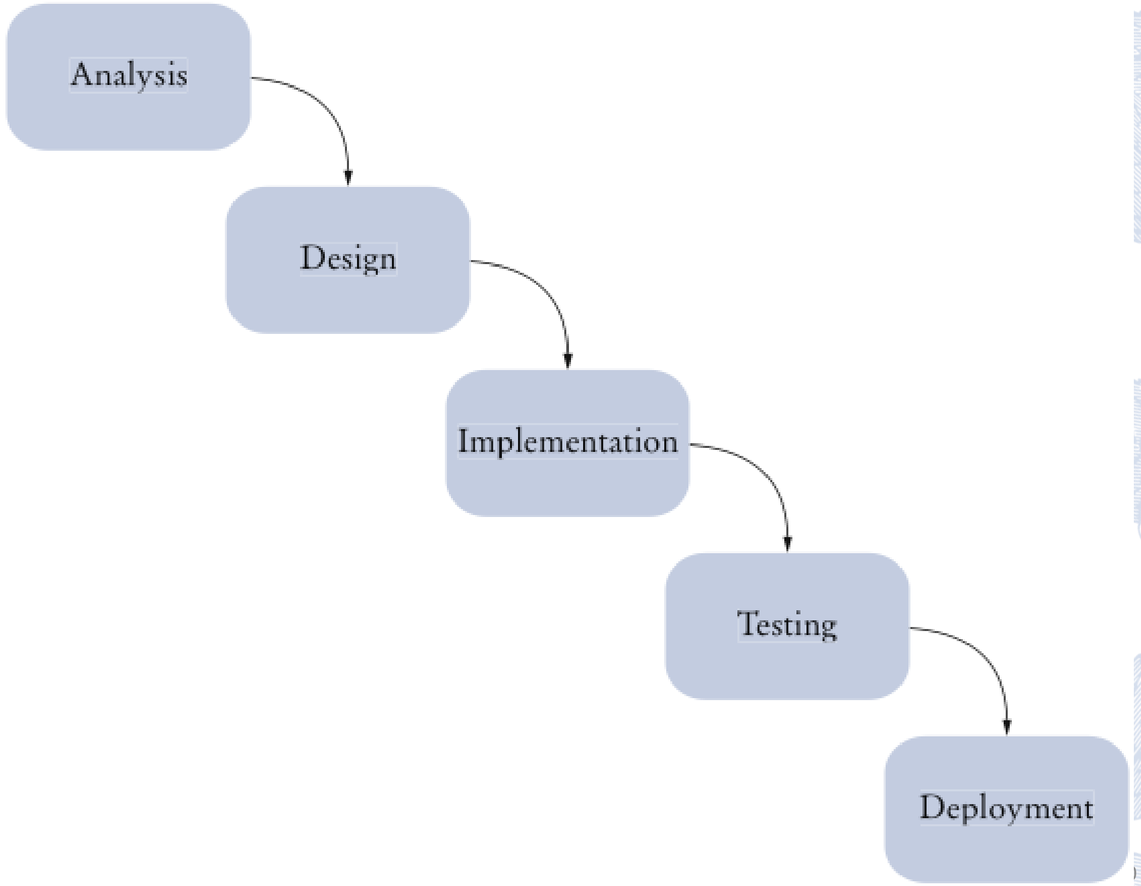


Figure 1 The Waterfall Model

Activity Levels in the Rational Unified Process

✓Development process methodology by the inventors of UML

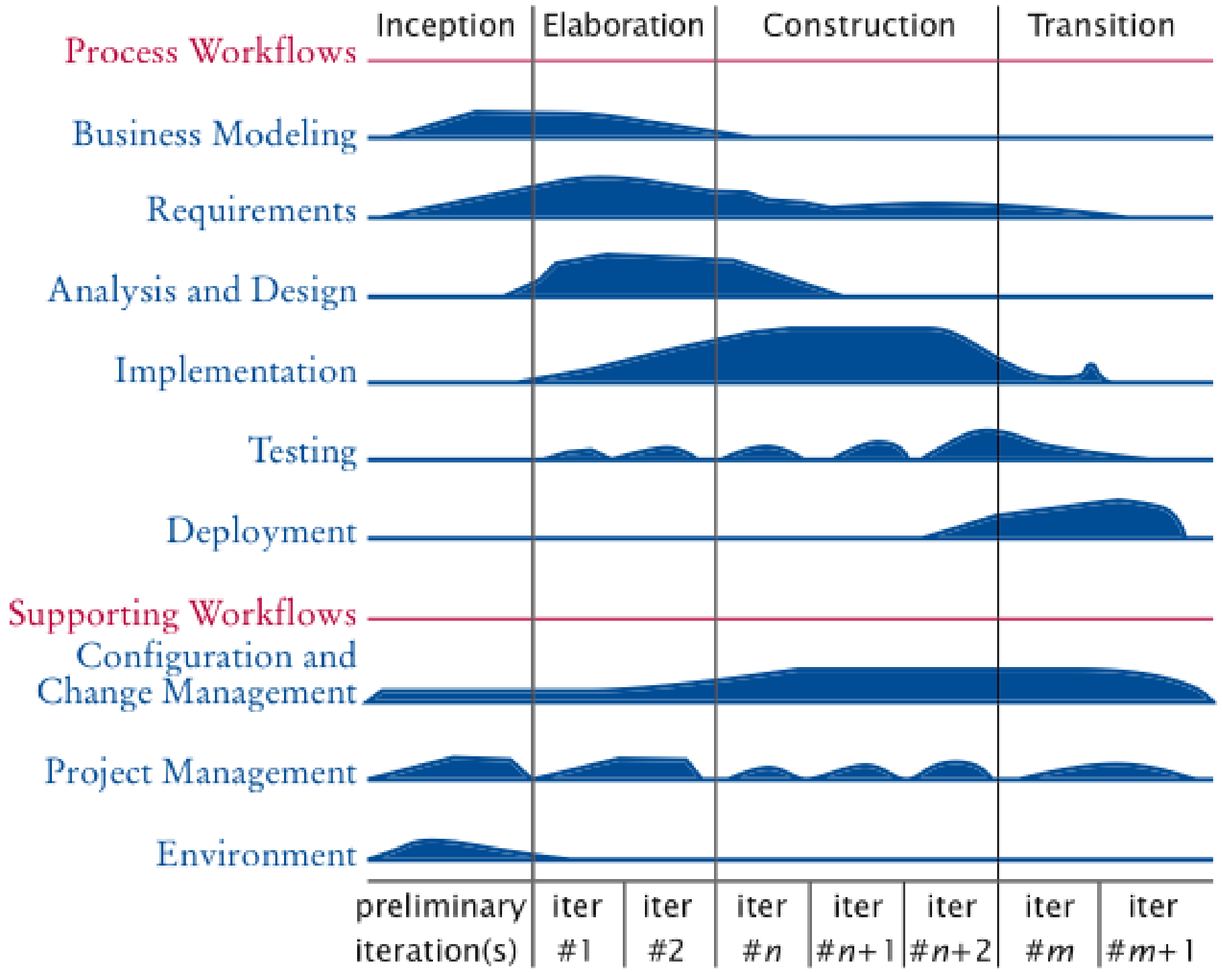


Figure 3 Activity Levels in the Rational Unified Process Methodology

Principles

- ✓Industry standard process model, initiated by Ericsson, then Objectory and Rational companies
- ✓Development of an OO system
- ✓Uses the UML notation throughout the process:
  - different views that are informal and not necessarily consistent.
- ✓Supports an iterative and incremental process
- ✓Decomposes a large process into controlled iterations (mini projects)

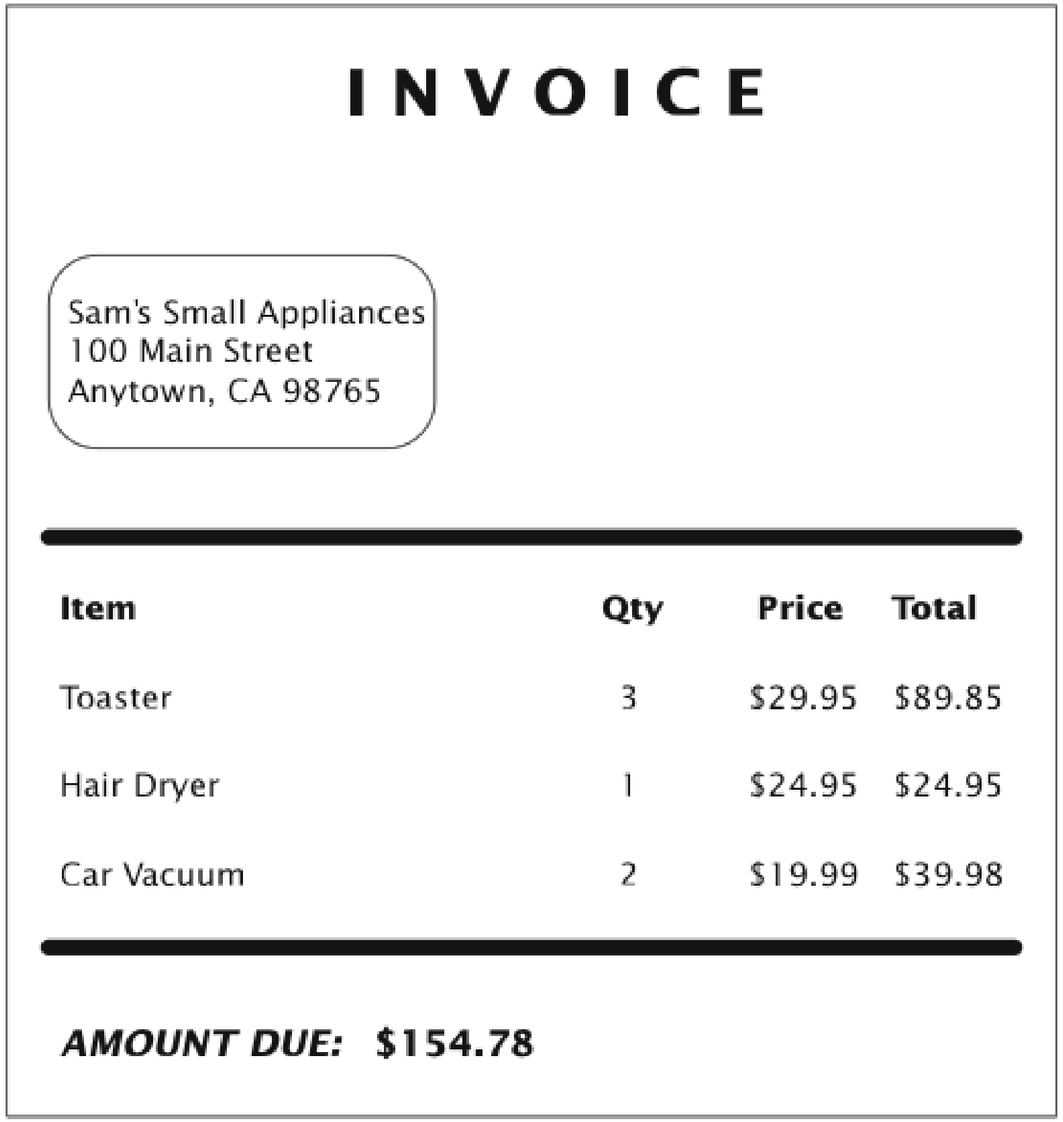
Object-Oriented Design

- ✓Discover classes
- ✓Determine responsibilities of each class
- ✓Describe relationships between the classes

Discovering Classes

- ✓A class represents some useful concept
- ✓Concrete entities: Bank accounts, ellipses, and products
- ✓Abstract concepts: Streams and windows
- ✓Find classes by looking for nouns in the task description
- ✓Define the behavior for each class
- ✓Find methods by looking for verbs in the task description

Example: Invoice



INVOICE

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

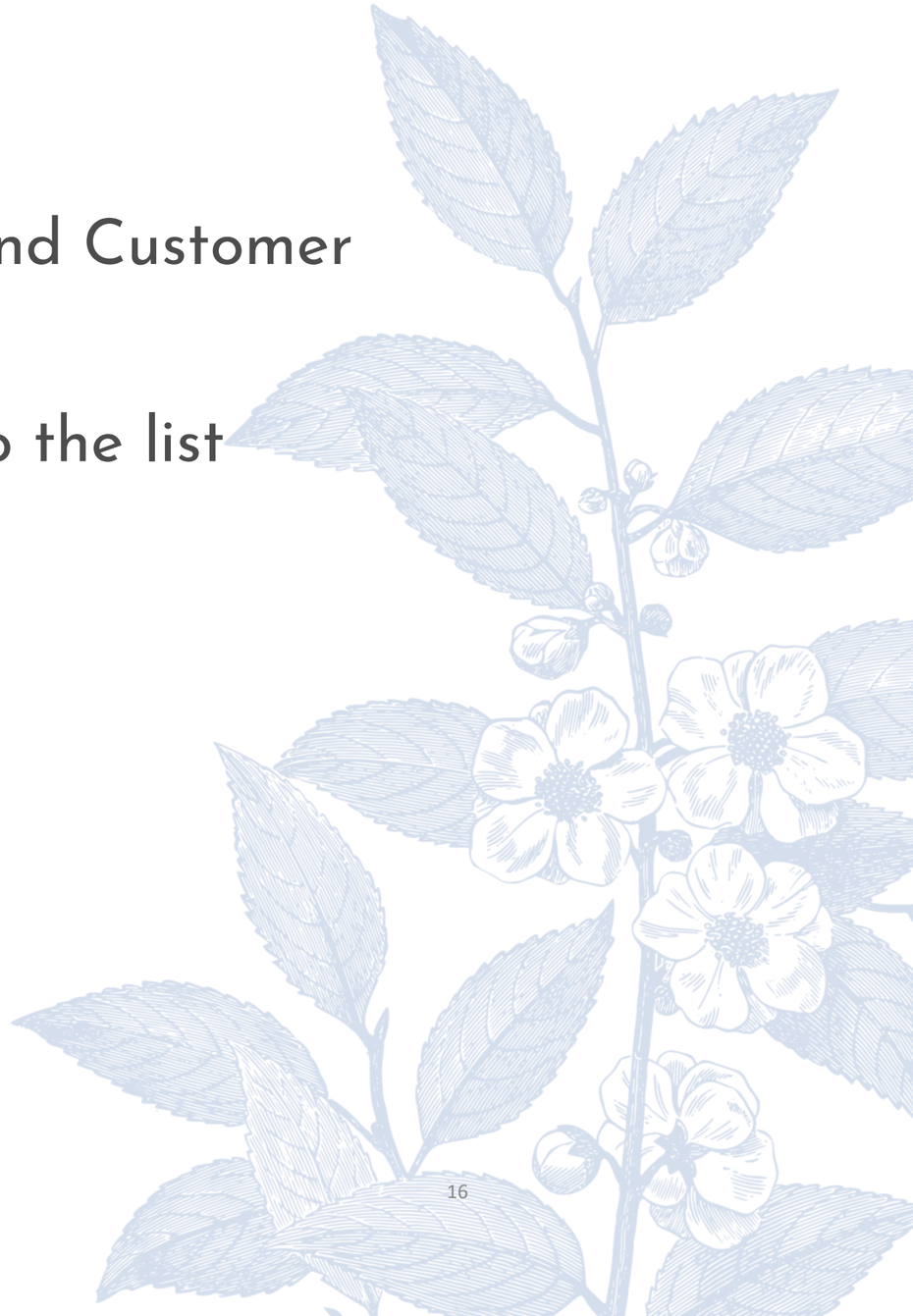
Item	Qty	Price	Total
Toaster	3	$29.95	$89.85
Hair Dryer	1	$24.95	$24.95
Car Vacuum	2	$19.99	$39.98

AMOUNT DUE: **$154.78**

Figure 4
An Invoice

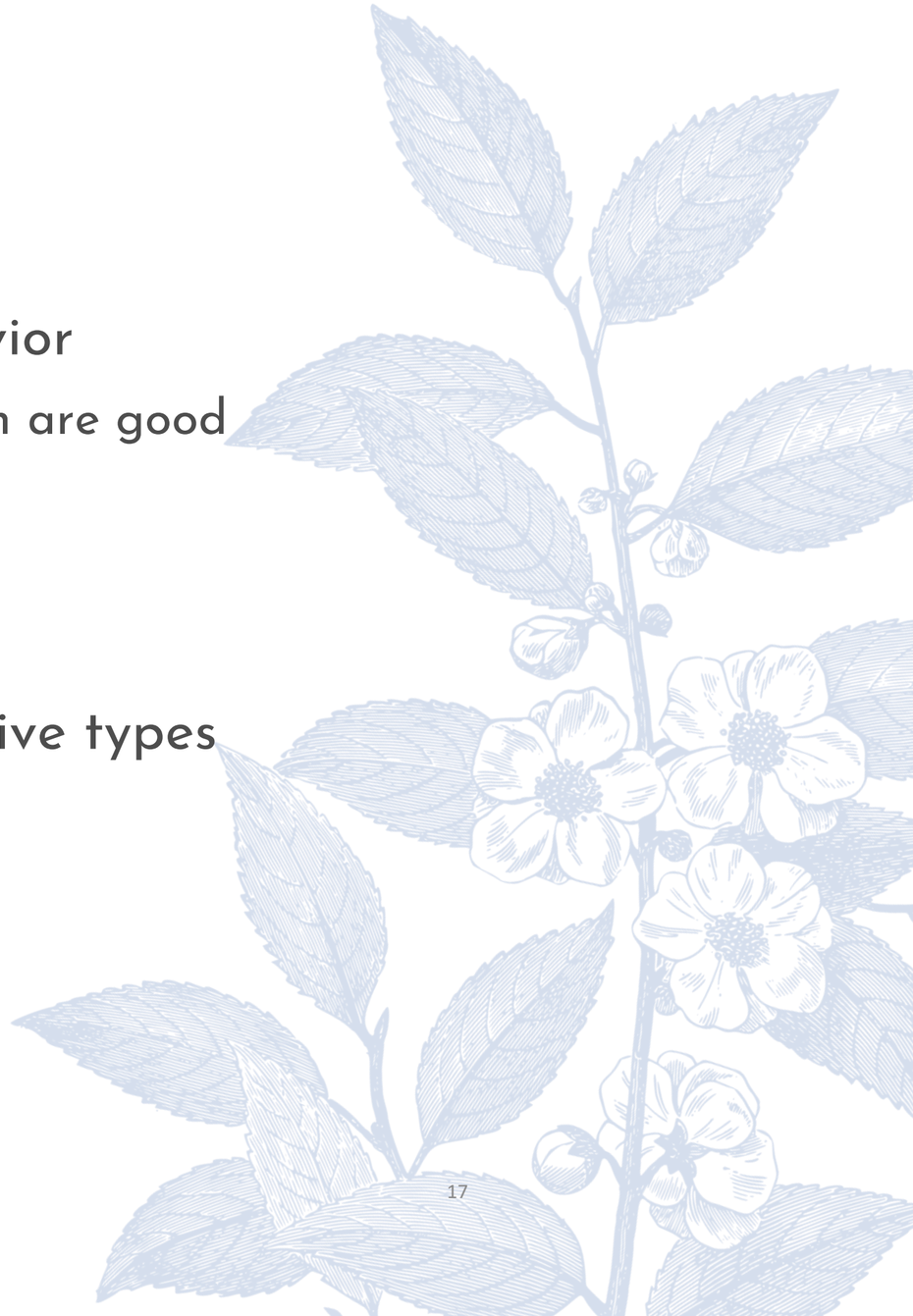
Example: Invoice

- ✓Classes that come to mind: Invoice, LineItem, and Customer
- ✓Good idea to keep a list of candidate classes
- ✓Brainstorm, simply put all ideas for classes onto the list
- ✓You can cross not useful ones later



Finding Classes

- ✓Keep the following points in mind:
  - Class represents set of objects with the same behavior
    - —Entities with multiple occurrences in problem description are good candidates for objects
    - —Find out what they have in common
    - —Design classes to capture commonalities
  - Represent some entities as objects, others as primitive types
    - —Should we make a class Address or use a String?
  - Not all classes can be discovered in analysis phase
  - Some classes may already exist



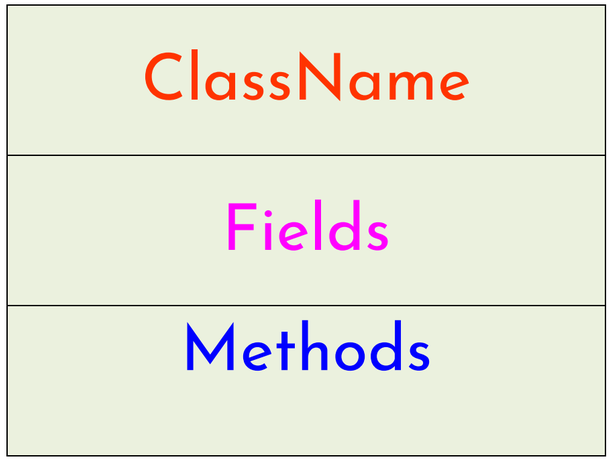
UML Diagram for a class

✓Unified Modeling Language (UML) provides a set of standard diagrams for graphically depicting object-oriented systems.

Class name goes here →

Fields are listed here →

Methods are listed here →

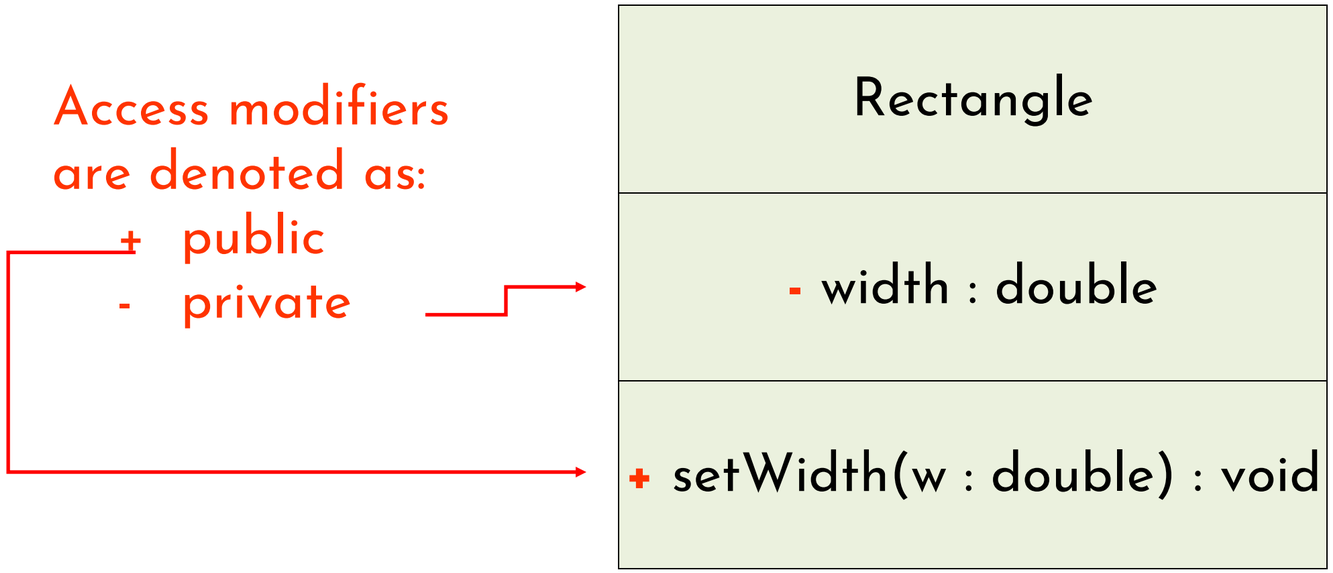


UML Data Type and Parameter Notation

- ✓UML diagrams are language independent.
- ✓UML diagrams use an independent notation to show return types, access modifiers, etc.

Access modifiers are denoted as:

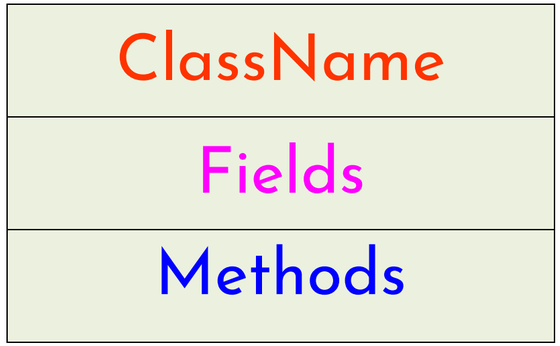
+ public
- private



Converting the UML Diagram to Code

- ✓Putting all of this information together, a Java class file can be built easily using the UML diagram.
- ✓The UML diagram parts match the Java class file structure.

```
class ClassName
{
  Fields
  Methods
}
```



Converting the UML Diagram to Code

The structure of the class can be compiled and tested without having bodies for the methods. Just be sure to put in dummy return values for methods that have a return type other than void.

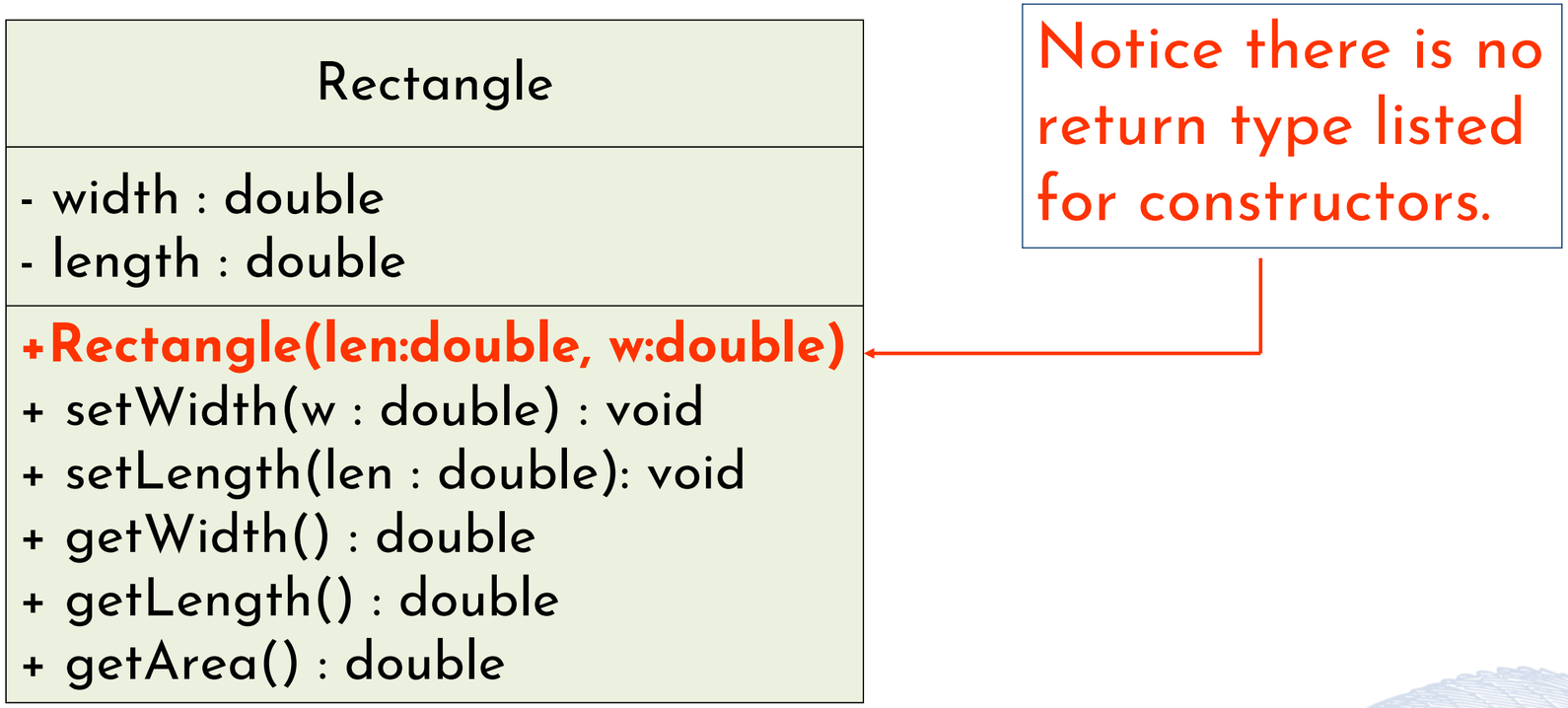
Rectangle
- width : double - length : double
+ setWidth(w : double) : void + setLength(len : double): void + getWidth() : double + getLength() : double + getArea() : double

```
public class Rectangle {
   private double width;
   private double length;

   public void setWidth(double w){
   }
   public void setLength(double len){
   }
   public double getWidth(){
        return 0.0;
   }
   public double getLength(){
        return 0.0;
   }
   public double getArea()   {
        return 0.0;
   }
}
```

Constructors in UML

✓In UML, the most common way constructors are defined is:



Notice there is no return type listed for constructors.

Relationships Between Classes

- ✓Association
- ✓Aggregation
- ✓Composition
- ✓Dependency

Relationships Between Classes

- ✓Association: More general relationship between classes
- ✓Use early in the design phase
- ✓A class is associated with another if you can navigate from objects of one class to objects of the other
- ✓Given a Bank object, you can navigate to Customer objects

An Association Relationship

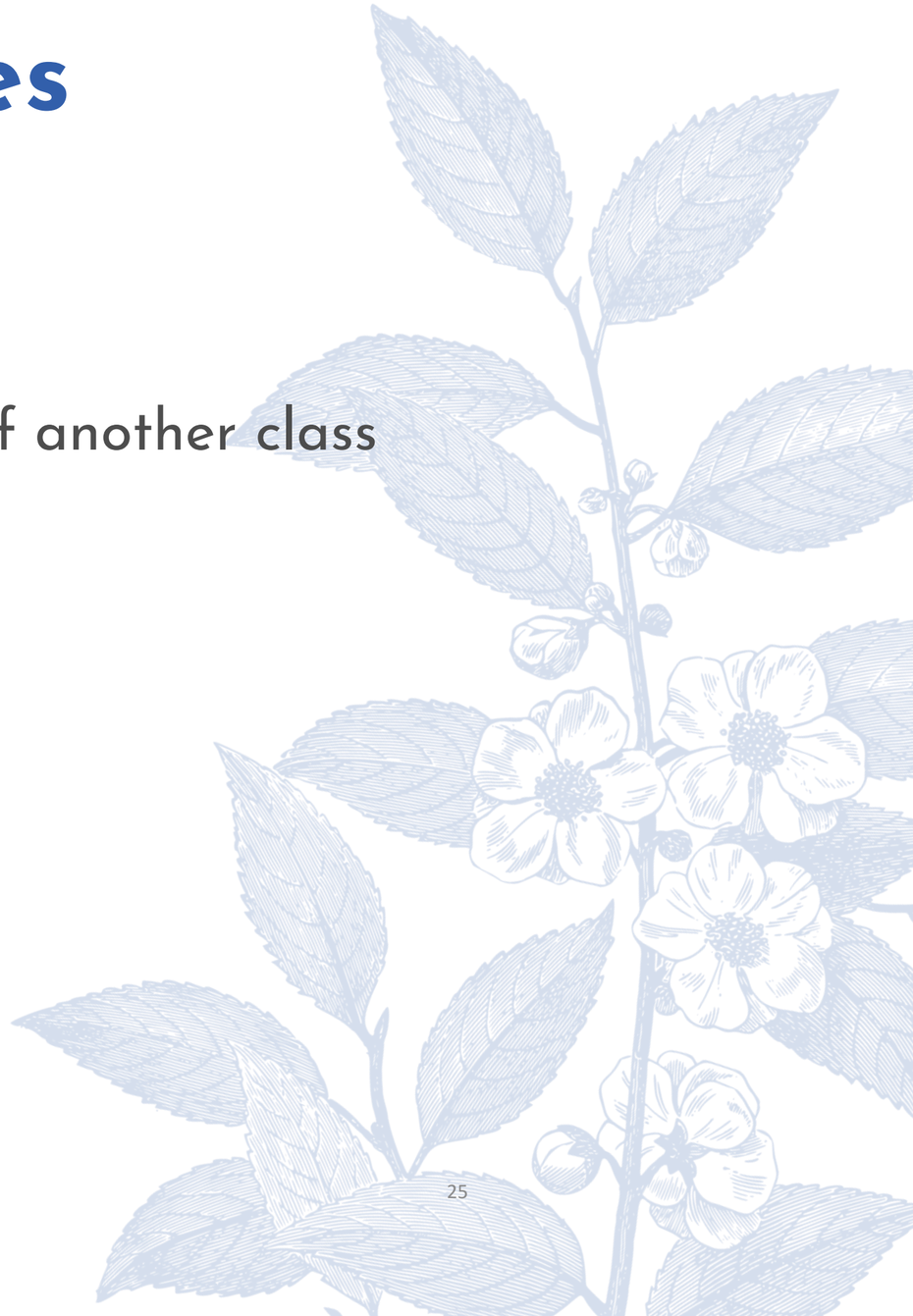
Relationships Between Classes

Aggregation

- ✓Has-a relationship
- ✓Objects of one class contain references to objects of another class
- ✓Use an instance variable
  - A tire has a circle as its boundary:

```
class Tire
{
   ...
   private String rating;
   private Circle boundary;
}
```

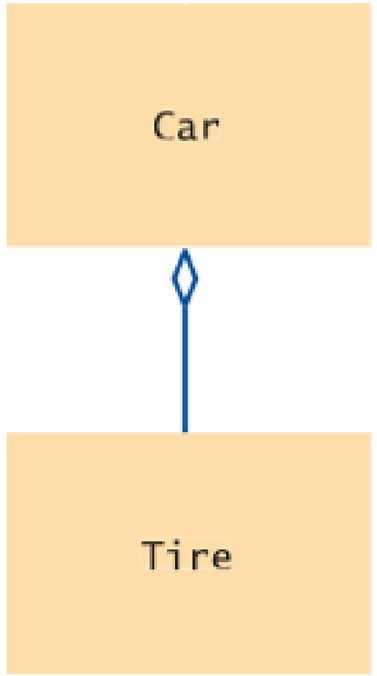
  - Every car has a tire (in fact, it has four)



Relationships Between Classes

Example

```
class Car
{
   ...
   private Tire[] tires;
}
```



Relationships Between Classes

Composition

- ✓An aggregation relationship where the aggregated objects do not have an existence independent of the containing object.
- ✓For example, composition models the relationship between a bank and its accounts. If a bank closes, the account objects cease to exist as well.

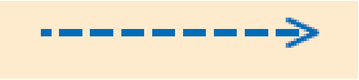
A Composition Relationship

Relationships Between Classes

Dependency

- ✓Uses relationship
  - Example: Many of our applications depend on the Scanner class to read input
- ✓Aggregation is a stronger form of dependency
- ✓Use aggregation to remember another object between method calls

UML Relationship Symbols

Relationship	Symbol	Line Style	Arrow Tip
Composition		Filled-in	Diamond
Interface Implementation		Dotted	Triangle
Aggregation		Solid	Diamond
Dependency		Dotted	Open
Association		Solid	Line

Multiplicities

- ✓any number (zero or more): *
- ✓one or more: 1..*
- ✓zero or one: 0..1
- ✓exactly one: 1

An Aggregation Relationship with Multiplicities

Five-Part Development Process

- ✓Gather requirements
- ✓Find classes, responsibilities, and collaborators
- ✓Use UML diagrams to record class relationships
- ✓Use javadoc to document method behavior
- ✓Implement your program