Designing Classes

When starting a software project, one of the first steps is to determine requirements. Next comes designing, which for object-oriented languages involves turning requirements into classes.

Manager:		Recorder:	
Presenter:		Reflector:	

Content Learning Objectives

After completing this activity, students should be able to:

- Explain the differences between classes, objects, and their attributes.
- Design a new class (in UML syntax) based on a general description.
- Draw a UML diagram that consists of multiple interrelated classes.

Process Skill Goals

During the activity, students should make progress toward:

- Identifying attributes and data types that model a real-world object. (Problem Solving)

Facilitation Notes

Point out that classes can be used to represent objects in the real world (such as a picture, animal, person, or car). They can also be used to represent abstract concepts (such as a color, word, or Scanner, or StringBuilder).

When designing classes for the first time, it might be easier to consider objects in the real world. Identifying appropriate data fields for real-world objects is more straightforward (less abstract).

Key questions: #3, #10, #11

Model 1 Repair Shop

You have been asked to design software for a local automobile repair shop:

> *Wrench Craft, owned by Dean George, is an independent automobile repair shop in Harrisonburg, VA, specializing in repair of Asian import vehicles such as Honda, Toyota, Subaru, Nissan, Isuzu, Mitsubishi, Hyundai, and Mazda. If you'd like to arrange a time for us to service your car, just give us a call and we'll set you up with an appointment to get the work done. What could be more simple?*
>
> Source: wrenchcraft.com

Among other things, the software needs to keep track of ***customers*** (e.g., name, address, phone number), ***cars*** (e.g., make, model, year), and ***invoices*** (e.g., parts, labor, notes).

Questions (15 min) Start time:

1. Identify all nouns in the description above. Don't worry about pronouns. List each noun only once (not every time it appears). Feel free to use an online tool like parts-of-speech.info.

software, automobile, repair, shop, Wrench, Craft, Dean, George, Harrisonburg, VA, import, vehicles, Honda, Toyota, Subaru, Nissan, Isuzu, Mitsubishi, Hyundai, Mazda, time, car(s), call, appointment, work, Source, wrenchcraft.com, track, customers, name, address, phone, number, make, model, year, invoices, parts, labor, notes

2. Discuss which nouns are more likely to be relevant for designing software. For each relevant noun, determine whether it represents a class, an object, or an attribute:

a) Classes: Shop, Appointment, Customer, Car, Invoice, . . .

b) Objects: Dean George, Harrisonburg, Honda, Toyota, . . .

c) Attributes: name, address, phone number, make, model, year, parts, labor, notes

3. Based on your discussion for #2, write a general definition for the following concepts:

a) Classes: General categories of real-world things

b) Objects: Examples (instances) of a particular category

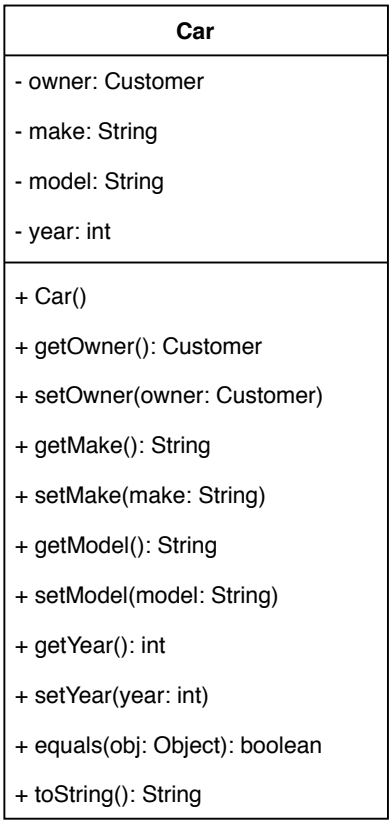
c) Attributes: Characteristics (or values) of a real-world thing

4. What other classes, objects, and attributes (not included in the description above) might be relevant for this software project?

Appointment (e.g., date, time, type)
Inventory (e.g., part, quantity)
Parts (e.g., number, name, cost)

Model 2 Class Diagram

Recall that a UML class diagram summarizes the attributes and methods of a class. In the repair shop example, a `Car` class might look something like this:



Questions (15 min)

Start time:

5. How many _____ are in the diagram above?

a) attributes 4

b) constructors 1

c) getters 4

d) setters 4

6. What is the type of the `owner` attribute? Is it a primitive or reference type?

It is a `Customer`, or more specifically, a reference to a `Customer` object.

7. Explain how this design allows multiple cars to be owned by the same customer.

Multiple `Car` objects can reference the same `Customer` object in memory.

8. List three attributes that would be appropriate for the `Customer` class.

Variable Name	Data Type	Example Value
name	String	"Jonathan Alger"
address	String	"800 S Main St ..."
phone	String	"(540) 568-6211"

9. Rewrite the attributes from the previous question in UML format.

- name: String
- address: String
- phone: String

10. For each attribute, define a `get` method. Write your answer in UML format.

+ getName(): String
+ getAddress(): String
+ getModel(): String

11. For each attribute, define a `set` method. Write your answer in UML format.

+ setName(String name)
+ setAddress(String address)
+ setModel(String model)

Optional Questions

12. What rules might be implemented in the `set` methods to ensure that only valid attribute values are stored? *Example: The customer's name should contain only letters and spaces.*

Variable Name	Validation Rules
name	no special characters, max length (e.g., of 50)
address	use an API like Google Maps to ensure it exists
phone	must be 10-digit number; use parens and dashes

13. Based on the attributes you defined, how could you determine whether two `Customer` objects represent the same customer?

Answers will vary. It's probably good enough to compare the name and phone number. But that would require names and numbers to be stored in a consistent format. A more reliable approach would be to define a unique customer number.

14. In Model 2, what is the parameter name and type for the `equals` method? What version of `equals` is this method overriding?

The name is `obj`, and the type is `Object`. The `Car` class is overriding the default version of `equals` defined in the `Ojbect` class.

Exercise: Bob's Grocery Mart

Bob's Grocery Mart has 73 locations distributed across 12 States. Bob would like to increase the profitability of his stores by improving cashier scheduling. If too few cashiers are scheduled, customers become frustrated and may not return to the store. With too many cashiers, Bob pays more wages than necessary. You have been tasked with writing a checkout simulation program that will be used to analyze the impact of increasing or decreasing the number of cashiers.

Specification

Note: You will only be designing—not implementing—this specification today.

Your program should simulate the activity of any number of staffed check-out aisles in a Bob's Grocery Mart. The simulation will proceed in one-second time steps. It takes between one and five seconds for a cashier to process a single object. During each one-second interval there is a 0.5% chance that a new customer will finish shopping and enter a checkout aisle.

Customers purchase between 1–100 items. If there are any empty aisles, customers choose one arbitrarily. If all aisles are occupied, customers choose the aisle with the shortest line.

The program should simulate a single eight-hour shift. At the end of the simulation, the program should display:

- The total number of customers served.
- The average wait time across all customers.
- The maximum wait time for any individual customer.

Design Steps (30 min) **Start time:**

a) Highlight all of the nouns from the program specification above. Circle entries that are good candidates for classes, and cross out entries that are duplicates or do not need to be represented in the program. Any remaining highlighted words should represent entities that need to be represented in the program but should not be classes.

b) On the next page, draw a largish UML box for each of the classes that you selected in the previous step. In the upper part of the box, list an appropriate set of attributes along with their types. Be sure to think about *persistent values* that constitute the state of an object vs. *computed values* that make more sense as local variables.

c) In the lower part of the box, list the necessary constructors for your classes, along with the required arguments. Does it make sense to have a default constructor? Are there classes that may need more than one constructor?

d) For each class, define appropriate methods including getters and setters, `equals` and `toString`, and others. Think about the parameters and return types for each method. Should all methods be public, or are there some that are needed only within the class?