Unit 3 Companion Sheet — Class Creation

Oct 8, 2026 · @Dan

How to use this sheet

Keep this open during Unit 3, not just before the test. Each lesson section follows Code.org CSA Alt 2025 lesson order and gives the lesson question, key vocabulary, the code pattern you will use, and a quick self-check.

Unit big idea: a class is a blueprint you write yourself. You decide its attributes (instance variables), how objects get built (constructors), how others read and change state (accessors and mutators), and what stays hidden (private).

Throughout this unit you will build the **Dessert** class and its subclass, then use the same skills in the **Store Management Project** and the **Burger Class FRQ**.

Lesson 1: Program Design

Lesson question: Why is it helpful to represent programs before we write them? How can you do this?

Term	Meaning
class	A blueprint that defines the attributes and behaviors (methods) of an object
object	A specific instance of a class
attribute	Data an object holds (its state)
behavior	What an object can do; written as methods
UML diagram	A visual model of a class: name, attributes, methods
inheritance	A subclass gets the attributes and methods of its superclass
superclass / subclass	The parent class / the child class that extends it

UML box layout (top to bottom: name, attributes, methods; `-` private, `+` public):

```
+------------------------------+
|           Dessert            |
+------------------------------+
| - name : String              |
| - price : double             |
+------------------------------+
| + Dessert(name, price)       |
| + getName() : String         |
| + setPrice(price) : void     |
+------------------------------+
         ^  (arrow points to superclass)
         |
+------------------------------+
|            Cake              |
+------------------------------+
```

Check yourself

- [ ] In a UML diagram, which section lists attributes? Which lists methods?
- [ ] Which way does the inheritance arrow point?
- [ ] Draw a UML box for a `Book` class with two attributes and two methods.

Lesson 2: Attributes

Lesson question: How can I represent the attributes an object should have in a class?

Term	Meaning
instance variable	A variable declared in the class (outside any method) that stores an object's attribute; each object gets its own copy
access modifier	A keyword (`public` or `private`) that controls which classes can use a variable or method
public	Accessible from any class
private	Accessible only inside the class where it is declared
encapsulation	Keeping data private inside a class and controlling access through methods

Term	Meaning
refactoring	Restructuring code to improve it without changing what it does

Pattern

```
public class FoodTruck {
    private String name;      // instance variables are private
    private int menuItems;
}

public class TacoTruck extends FoodTruck {
    private boolean hasSalsaBar;   // subclass adds only what's new
}
```

Rules to remember

- Instance variables are declared **private** (AP expectation).
- A subclass inherits the superclass's attributes; do not redeclare them.
- A subclass cannot directly use the superclass's private variables; it goes through public methods or `super(...)`.

Check yourself

- [ ] Why make instance variables private instead of public?
- [ ] Where in a class are instance variables declared?
- [ ] A `TacoTruck` needs a `name`. Should it declare its own `name` variable? Why or why not?

Lesson 3: The this Keyword

Lesson question: How can I reduce ambiguity and redundancies in my program?

Term	Meaning
constructor	A special method with the class's name and no return type that initializes a new object's instance variables
parameterized constructor	A constructor that takes parameters to set the starting values

Term	Meaning
no-argument constructor	A constructor with no parameters that sets default values
local variable	A variable declared inside a method or constructor (including parameters)
scope	Where in the code a variable can be used; a local variable's scope ends at the closing brace of its method
this	A keyword that refers to the current object
super	A keyword used in a subclass to call the superclass's constructor or methods

Pattern

```
public Dessert(String name, double price) {
    this.name = name;     // this.name = instance variable
    this.price = price;   // price alone = the parameter (local)
}

public Cake(String name, double price, int layers) {
    super(name, price);   // must be the first line
    this.layers = layers;
}
```

Watch out

- If a parameter has the same name as an instance variable, the parameter **shadows** it. Without `this.`, `name = name;` just assigns the parameter to itself.
- A constructor has **no** return type, not even `void`.

Check yourself

- [ ] What does `name = name;` do inside a constructor whose parameter is `name`?
- [ ] Can a local variable from one constructor be used in another method?
- [ ] Where must `super(...)` go in a subclass constructor?

Lesson 4: Accessor Methods

Lesson question: How can I access the values of an object's instance variables?

Term	Meaning
accessor method	A method that returns the value of an instance variable without changing it (a "getter")
return type	The data type of the value a method sends back
return statement	Ends the method and sends a value back to the caller
method signature	The method's name and parameter list

Pattern

```
public String getName() {
    return name;
}

public double getPrice() {
    return price;
}
```

Rules to remember

- Accessors are usually `public`, take **no parameters**, and have a return type that matches the variable.
- Only write an accessor if another class actually needs that value.
- A subclass can call the superclass's public accessors to reach private data.

Check yourself

- [ ] What return type does `getLayers()` need if `layers` is an `int`?
- [ ] Why might you choose **not** to write an accessor for an instance variable?
- [ ] Write the accessor for a `private boolean isFrozen;` variable.

Lesson 5: Mutator Methods

Lesson question: How can I change the values assigned to an object's instance variables after the object has been created?

Term	Meaning
mutator method	A method that changes the value of an instance variable (a "setter")
void	A return type meaning the method returns nothing
Scanner	A class in `java.util` that reads user input
nextLine() / nextInt() / nextDouble()	Scanner methods that read a line of text, an int, or a double
invalid value	A value that doesn't make sense for the attribute (for example, a negative price)

Pattern

```
public void setPrice(double price) {
    if (price >= 0) {          // guard against invalid values
        this.price = price;
    }
}
```

```
import java.util.Scanner;

Scanner input = new Scanner(System.in);
System.out.print("New price: ");
double newPrice = input.nextDouble();
myCake.setPrice(newPrice);
```

Rules to remember

- Mutators are usually `public void` and take **one parameter**.
- Not every variable should have a mutator. Something that should never change (like an ID) gets no setter.
- Use an `if` to reject invalid values before assigning.

Check yourself

- [ ] Why is a mutator's return type usually `void`?
- [ ] Name an attribute that should **not** have a mutator, and explain why.
- [ ] Write `setLayers(int layers)` so it only accepts values from 1 to 5.

Lesson 6: Static Variables and Methods

Lesson question: Why would I use static variables and methods?

Term	Meaning
static	A keyword meaning the variable or method belongs to the class, not to any one object
static variable (class variable)	One shared copy for all objects of the class
static method (class method)	A method called on the class name; it cannot use instance variables or `this`
helper method	A method that does a small, reusable task, often static
instance method	A non-static method; it needs an object and can use instance variables

Pattern

```
public class Dessert {
    private static int dessertCount = 0;   // shared by all Desserts
    private String name;

    public Dessert(String name) {
        this.name = name;
        dessertCount++;                     // every new object adds 1
    }

    public static int getDessertCount() {
        return dessertCount;
    }
}

int total = Dessert.getDessertCount();     // called on the class
```

Rules to remember

- Static methods **cannot** use instance variables or `this`.
- Instance methods **can** use static variables.
- You've already used static methods: `Math.random()`, `Math.abs()`.

Check yourself

- [ ] If 3 `Dessert` objects are made, what does `getDessertCount()` return?
- [ ] Why can't a static method use `this.name`?
- [ ] How is calling a static method different from calling an instance method?

Lesson 7: Object References as Parameters

Lesson question: What are the similarities and differences between passing primitives and object references as parameters?

Term	Meaning
primitive type	A basic data type (`int`, `double`, `boolean`) that stores its value directly
reference type	A type (any class, like `String` or `Dessert`) whose variable stores the object's address
object reference	The address that points to where an object is stored in memory
pass by value	Java copies the argument into the parameter; for objects, the **reference** is copied
user story	A short description of a feature from the user's point of view: "As a ___, I want ___ so that ___."

Primitive vs. object reference

	Primitive parameter	Object reference parameter
What gets copied	The value	The address

	Primitive parameter	Object reference parameter
Can the method change the caller's original?	No	Yes, by calling its mutators
Example	`double p = 5.0;`	`Dessert d = new Cake(...);`

```
public static void discount(Dessert d) {
    d.setPrice(d.getPrice() * 0.9);   // changes the ORIGINAL object
}
```

Watch out: if a method should not change the original object, don't call mutators on the parameter. Pass only the values it needs.

Project tie-in: write user stories for your project now, then update your Need to Knows.

Check yourself

- [ ] After `discount(myCake)`, has `myCake`'s price changed? Why?
- [ ] A method sets a primitive parameter `x = 0;`. Does the caller's variable change?
- [ ] Write one user story for your Store Management Project.

Lesson 8: Private Methods

Lesson question: How does making a method private change its functionality?

Term	Meaning
private method	A method that can only be called from inside its own class
public method	A method any class can call; part of the class's interface
procedural abstraction	Hiding the steps of a task inside a named method so callers use it without knowing the details

Pattern

```
public class Dessert {
    private double price;

    public double getPriceWithTax() {
        return price + calculateTax();   // OK: called inside the class
    }

    private double calculateTax() {      // helper hidden from other classes
        return price * 0.0725;
    }
}

// In another class:
// myCake.calculateTax();   // ERROR: calculateTax() is private
```

When to make a method private: it's a helper step other classes don't need, or calling it from outside could put the object in a bad state.

Project tie-in: start planning the Creative Coding with The Theater project: UML diagrams, algorithms, and your Project Planning Board.

Check yourself

- [ ] Can a subclass call its superclass's private method directly?
- [ ] Give one reason to make a method private.
- [ ] What error happens when another class calls a private method?

Lessons 9–13: Computing Impacts

These lessons are mostly discussion. Each has a Check for Understanding, so know the key terms and be ready to give one benefit and one harm.

Lesson	Lesson question	Key terms
9: AI-Generated Code	How can a programmer use AI effectively and ethically to develop code?	artificial intelligence (AI), bias, data set, prompt, prompt design

Lesson	Lesson question	Key terms
10: System Reliability	What issues and impacts result from software bugs, and what are strategies to maximize system reliability?	bug, system reliability, open source software, bug report, contributing code
11: Privacy and Security	What are the tradeoffs between the utility of data collection and safeguarding privacy and security?	data collection, privacy, security, personally identifiable information (PII), data breach
12: Impacts of Programs	What are the beneficial and harmful impacts of programs?	beneficial impact, harmful impact, deep fake, image manipulation, object detection
13: Intellectual Property	Are AI-generated images art?	intellectual property, copyright, AI-generated image, training data, permission

Quick reminders

- **Lesson 9:** AI-generated code can contain bugs or bias. Always read, test, and understand it before using it. A specific prompt (language, class names, requirements) gets better results.
- **Lesson 10:** Anyone can help open source software by reporting bugs or contributing fixes.
- **Lesson 11:** Apps collect data to be useful, but every piece collected is something that can be breached.
- **Lesson 12:** Deep fakes are hard to spot. This lesson has Investigate and Modify levels on image manipulation algorithms, so expect code, not just discussion.
- **Lesson 13:** AI image tools are often trained on artists' work without their knowledge or permission.

Check yourself

- [ ] Name one way bias can end up in an AI-generated program.
- [ ] Give one benefit and one harm of deep fakes.
- [ ] What is one tradeoff of an app collecting your location?

Store Management Project and Burger Class FRQ

Store Management Project: build a store management system for a business or service you care about, using every skill from this unit.

- [ ] UML diagram drawn for each class, including any superclass/subclass
- [ ] User stories written for the main features
- [ ] Private instance variables
- [ ] Parameterized constructor using `this` (and `super(...)` in subclasses)
- [ ] Accessor methods only where needed
- [ ] Mutator methods that reject invalid values
- [ ] At least one static variable or method used for a real reason
- [ ] At least one private helper method
- [ ] Scanner input from the user
- [ ] Comments explaining each class and method

Burger Class FRQ (Classes and Objects): write a full class from a written description.

1. **Decompose:** underline every attribute (→ instance variable) and every action (→ method).
2. **Annotate:** next to each method, write its return type and parameters.
3. **Write the header:** `public class Burger`.
4. **Declare** instance variables as `private`.
5. **Write the constructor:** same name as the class, no return type, assign every instance variable.
6. **Write each method:** match the exact name, return type, and parameters given.
7. **Validate:** check your answer against the Scoring Guidelines, point by point.

Common FRQ point losses: public instance variables, a return type on the constructor, a missing `return`, or a changed method name.

Unit 3 Vocabulary Bank

All terms in lesson order, for quick review.

Lesson	Terms
1	class, object, attribute, behavior, UML diagram, inheritance, superclass, subclass
2	instance variable, access modifier, public, private, encapsulation, refactoring
3	constructor, parameterized constructor, no-argument constructor, local variable, scope, this, super
4	accessor method, return type, return statement, method signature
5	mutator method, void, Scanner, nextLine(), nextInt(), nextDouble(), invalid value
6	static, static variable (class variable), static method (class method), helper method, instance method
7	primitive type, reference type, object reference, pass by value, user story
8	private method, public method, procedural abstraction
9	artificial intelligence (AI), bias, data set, prompt, prompt design
10	bug, system reliability, open source software, bug report
11	data collection, privacy, security, PII, data breach
12	beneficial impact, harmful impact, deep fake, image manipulation, object detection
13	intellectual property, copyright, AI-generated image, training data