AP CSA · UNIT 3

# Class Creation

A class is a blueprint you write yourself. This unit, you design it: what it stores, how it is built, and who can touch it.

## BUILD THE CLASS · LESSONS 1–8

1. **Program Design** — Plan it with a UML diagram
2. **Attributes** — Private instance variables
3. **The this Keyword** — Constructors, scope, this
4. **Accessor Methods** — Read state with getters
5. **Mutator Methods** — Change state with setters
6. **Static Variables and Methods** — Shared by the whole class
7. **Object References as Parameters** — Copy a value or an address
8. **Private Methods** — Hide helper steps

## THINK ABOUT IMPACT · LESSONS 9–13

9. **AI-Generated Code**
10. **System Reliability**
11. **Privacy and Security**
12. **Impacts of Programs**
13. **Intellectual Property**

## SHOW WHAT YOU KNOW

**Store Management Project**
Model a business you care about

**Burger Class FRQ**
Write a full class from a prompt

LESSONS 2–5

# Anatomy of a Class

Every class you write in this unit has the same four parts, in this order.

| | |
|---|---|
| `public class Dessert {` | **Class header**<br>Name starts with a capital letter |
| `private String name;`<br>`private double price;` | 1 · LESSON 2<br>**Instance variables**<br>The object's attributes. Always private. |
| `public Dessert(String name,`<br>`double price) {`<br>`this.name = name;`<br>`this.price = price;`<br>`}` | 2 · LESSON 3<br>**Constructor**<br>Same name as the class. No return type. Sets every instance variable. |
| `public String getName() {`<br>`return name;`<br>`}` | 3 · LESSON 4<br>**Accessor (getter)**<br>No parameters. Returns a value. Changes nothing. |
| `public void setPrice(double price) {`<br>`if (price >= 0) {`<br>`this.price = price;`<br>`}`<br>`}` | 4 · LESSON 5<br>**Mutator (setter)**<br>Returns void. Takes one parameter. Rejects invalid values. |
| `}` | Closing brace ends the class |

**Getter: returns, no parameter**

**Setter: void, one parameter**

LESSON 3

# this and Scope

`this` means "the current object." Use it when a parameter has the same name as an instance variable.

## WHERE EACH VARIABLE LIVES

```
class Dessert                                   scope: the whole class

  name   price    instance variables

  Dessert(String name, double price)            scope: only inside { }

    name   price    parameters = local variables
```

Inside the constructor, plain `name` means the parameter. It shadows the instance variable.

### ✕ Does nothing

```
name = name;
price = price;
```

The parameter is assigned to itself. The object stays empty.

### ✓ Works

```
this.name = name;
this.price = price;
```

Left side: the object's variable. Right side: the parameter.

**`super(...)`** In a subclass constructor, call the superclass constructor first. It must be the very first line.

LESSONS 2 AND 8

# Public vs Private

**Encapsulation:** keep data locked inside the class. Other classes get in only through public methods.

INSIDE class Dessert

🔒 `private double price;`

🔒 `private String name;`

🔒 `private double calcTax()`

Only code in this class can use these.

**PUBLIC DOOR**

`+ getPrice()`

`+ setPrice(p)`

`+ getPriceWithTax()`

OTHER CLASSES

Can call the public methods. Cannot touch anything locked.

In another class

```
cake.getPrice();    // OK
cake.price = 2;     // ERROR
cake.calcTax();     // ERROR
```

Make a method private when...

It is a helper step other classes don't need.

Calling it from outside could break the object.

**–**

private in UML

**+**

public in UML

Subclasses can't reach the superclass's private parts either.

LESSON 6

# Static vs Instance

`static` belongs to the class. Everything else belongs to each object.




Each object has its own name. All three share one count.

| | static | instance |
|---|---|---|
| Belongs to | the class | each object |
| Called on | `Dessert.getCount()` | `cake.getName()` |
| Uses instance variables or this? | **No** | **Yes** |
| You've seen | `Math.random()` | `painter.move()` |

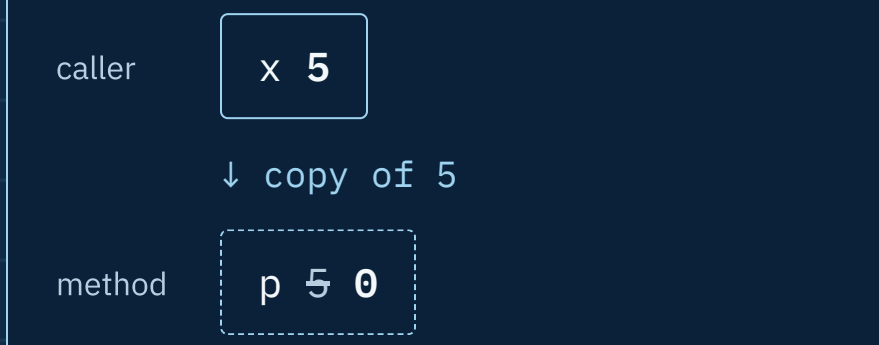
LESSON 7

# Passing Parameters

Java always passes a copy. What gets copied decides whether the original can change.

PRIMITIVE · int, double, boolean

## Copies the value




Caller's x is still **5**. The original can't change.

OBJECT REFERENCE · any class

## Copies the address




The method's mutator changed the **original** object.

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

ALSO IN LESSON 7 · USER STORY

As a [type of user], I want [a feature] so that [a reason].

LESSONS 9–13

# Computing Impacts

For each lesson, be ready to name one benefit and one harm.

## 9 AI-Generated Code

AI code can carry bugs and bias. Read it, test it, understand it. Specific prompts get better code.

`AI · bias · data set · prompt`

## 10 System Reliability

Bugs have real costs. Anyone can help open source software by reporting bugs or contributing fixes.

`bug · system reliability · open source · bug report`

## 11 Privacy and Security

Collecting data makes apps useful. Every piece collected is also something that can be breached.

`data collection · privacy · security · PII · data breach`

## 12 Impacts of Programs

Deep fakes are hard to spot. You'll also code image manipulation algorithms.

`beneficial · harmful · deep fake · image manipulation`

## 13 Intellectual Property

Are AI-generated images art? Many tools train on artists' work without permission.

`intellectual property · copyright · training data`

**BENEFIT**
Who does this help, and how?

**HARM**
Who could this hurt, and how?

BURGER CLASS FRQ

# Write a Class in 7 Steps

1. **Decompose.** Underline attributes → instance variables. Circle actions → methods.
2. **Annotate.** Write each method's return type and parameters.
3. **Header.** `public class Burger {`
4. **Declare** every instance variable `private`.
5. **Constructor.** Class name, no return type, set every variable.
6. **Methods.** Match the exact name, return type and parameters.
7. **Validate** against the Scoring Guidelines, point by point.

## WHERE POINTS GET LOST

- Public instance variables
- A return type on the constructor
- A missing return
- A changed method name

## STORE MANAGEMENT PROJECT CHECKLIST

- ☐ UML diagram for each class
- ☐ User stories
- ☐ Private instance variables
- ☐ Constructor with this / super
- ☐ Accessors where needed
- ☐ Mutators that reject bad values
- ☐ A static variable or method
- ☐ A private helper method
- ☐ Scanner input
- ☐ Comments on every method