Downloading Eclipse IDE.

Step 1- https://www.eclipse.org/downloads/

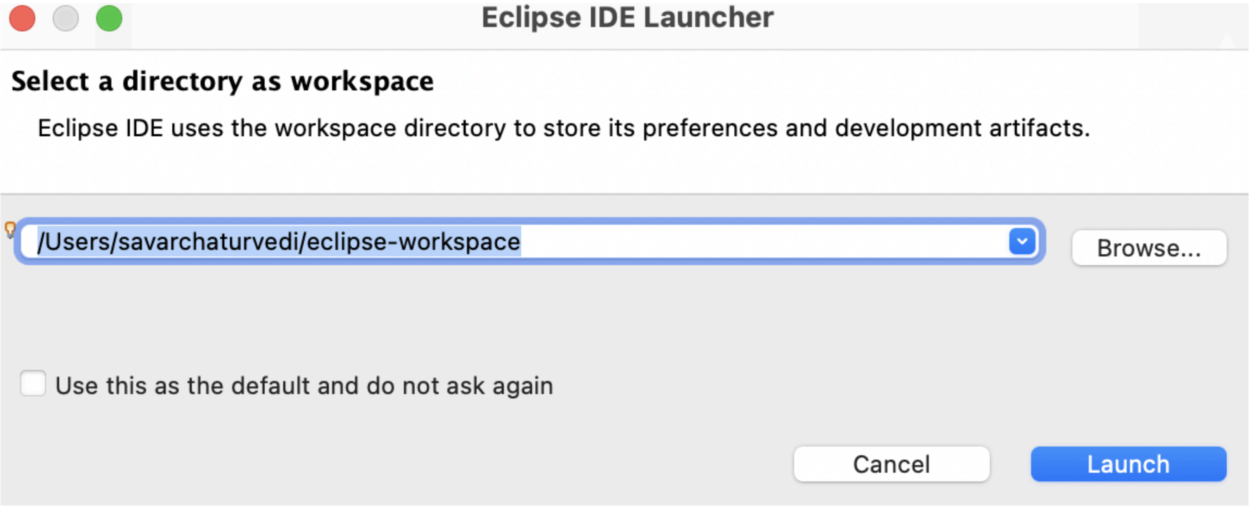
Step2 - Under **Install your favorite desktop IDE packages -> click download Packages.**

Step 3- choose the variant based on the Operating system of your computer.

Step 4 - Click Download.

Step 5 - Let the download complete. Open eclipse

Step 6- Select default directory



Eclipse IDE Launcher

Select a directory as workspace

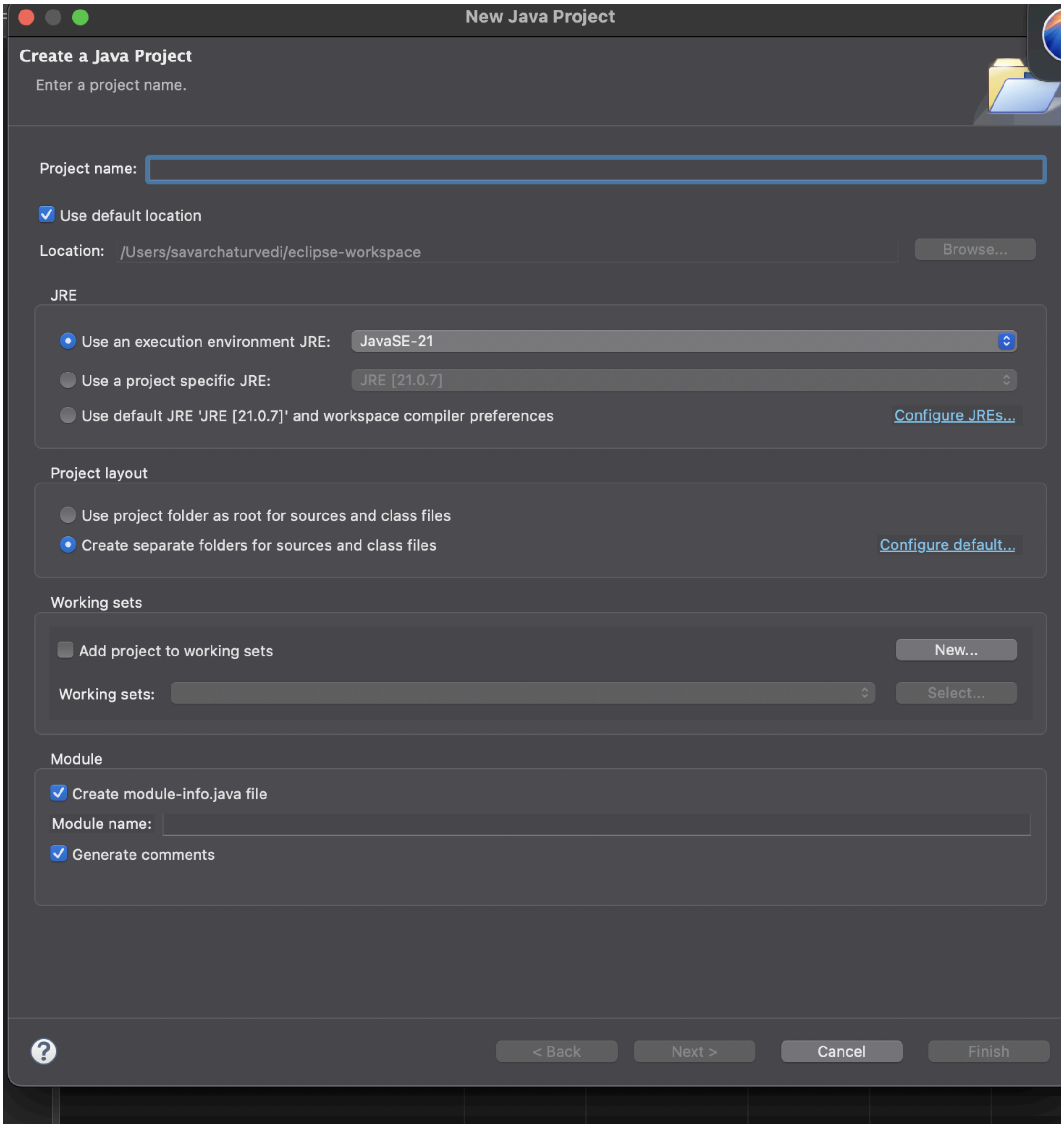
Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

/Users/savarchaturvedi/eclipse-workspace

□ Use this as the default and do not ask again

Step 7- Click Launch

Step 8 - create first java project



Create a Java Project

Enter a project name.

Project name:

☑ Use default location

Location: /Users/savarchaturvedi/eclipse-workspace

JRE

● Use an execution environment JRE: JavaSE-21

○ Use a project specific JRE: JRE [21.0.7]

○ Use default JRE 'JRE [21.0.7]' and workspace compiler preferences

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

Working sets

□ Add project to working sets

Working sets:

Module

☑ Create module-info.java file

Module name:

☑ Generate comments

Java Review-

Java is Object-Oriented

- Everything is written inside classes.
- Objects are created from classes.

Write Once, Run Anywhere

- Java code is converted into bytecode.
- Bytecode runs on the JVM (Java Virtual Machine), so the same program works on Windows, Mac, Linux.

Main Method is the Starting Point

```
public static void main(String[] args) {
   System.out.println("Hello World");
}
```

Variables and Data Types

Example:

```
 int age = 20;       // integer
double gpa = 3.5;   // decimal
String name = "Sam"; // text
boolean isOn = true; // true/false
```

Access Modifiers

Decide where something can be used:

public → anywhere

private → only inside the class

default → only in same package

protected → same package + child classes

Object-Oriented Programming (OOP) & Inheritance

What is OOP?

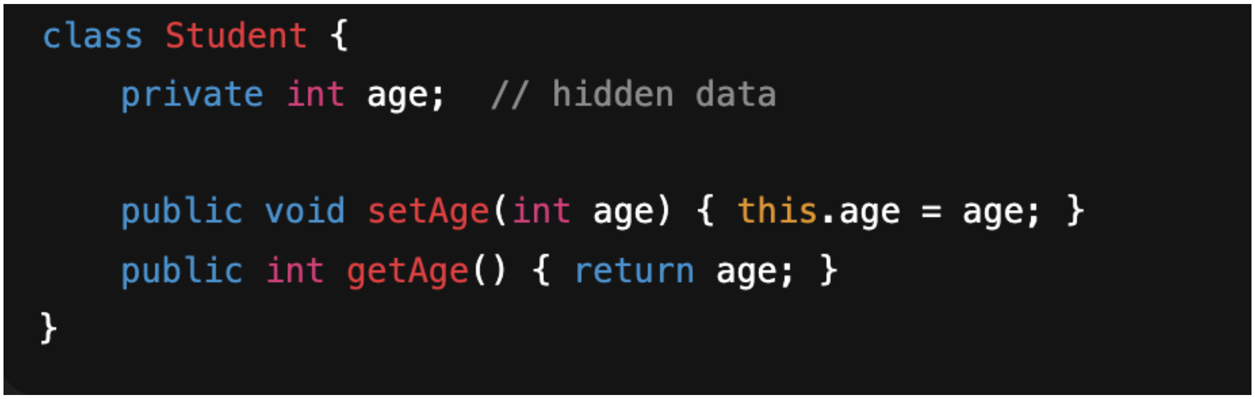
- OOP = **Object-Oriented Programming**.

- It organizes code into **classes** (blueprints) and **objects** (real things created from classes).

- Makes programs easier to **reuse, maintain, and scale**.

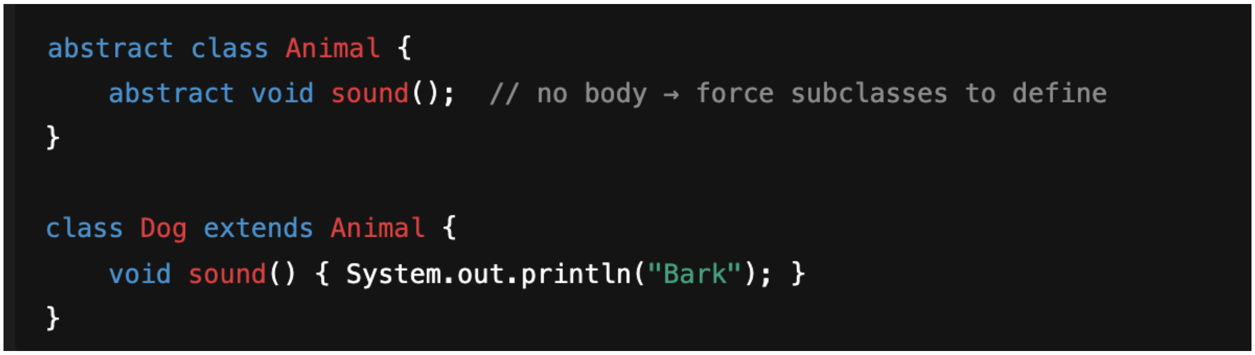
Four Pillars of OOP

1. **Encapsulation** → Hiding details inside a class (like data in private fields).



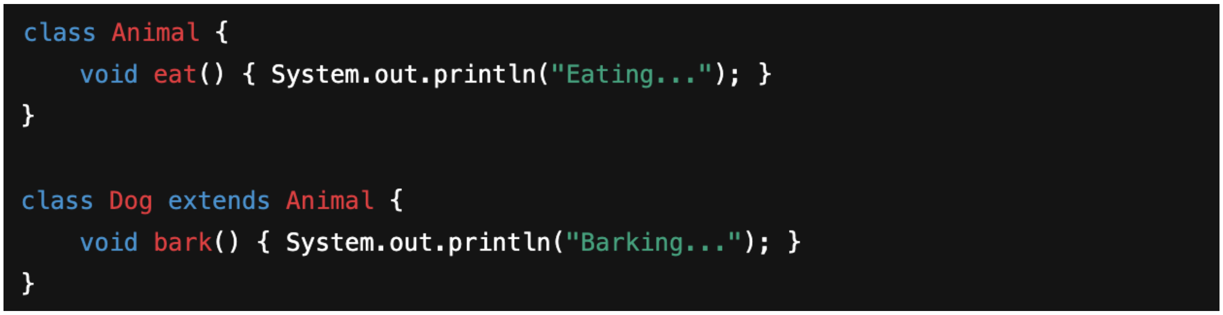
```java
class Student {
    private int age;  // hidden data

    public void setAge(int age) { this.age = age; }
    public int getAge() { return age; }
}
```

2. **Abstraction** → Hiding **implementation details**, showing only what's necessary.



```java
abstract class Animal {
    abstract void sound();  // no body → force subclasses to define
}

class Dog extends Animal {
    void sound() { System.out.println("Bark"); }
}
```

3. **Inheritance** → One class can reuse another class's properties/methods.



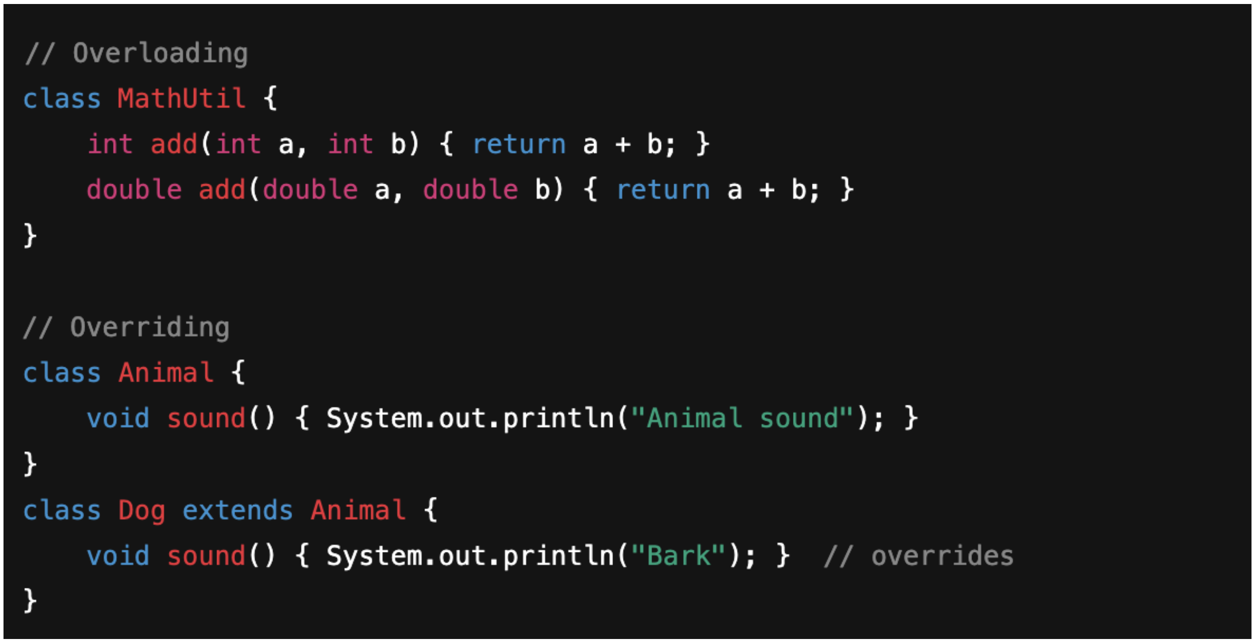
```
class Animal {
    void eat() { System.out.println("Eating..."); }
}

class Dog extends Animal {
    void bark() { System.out.println("Barking..."); }
}
```

4. **Polymorphism → Many Forms**

The same method behaves differently depending on the object.

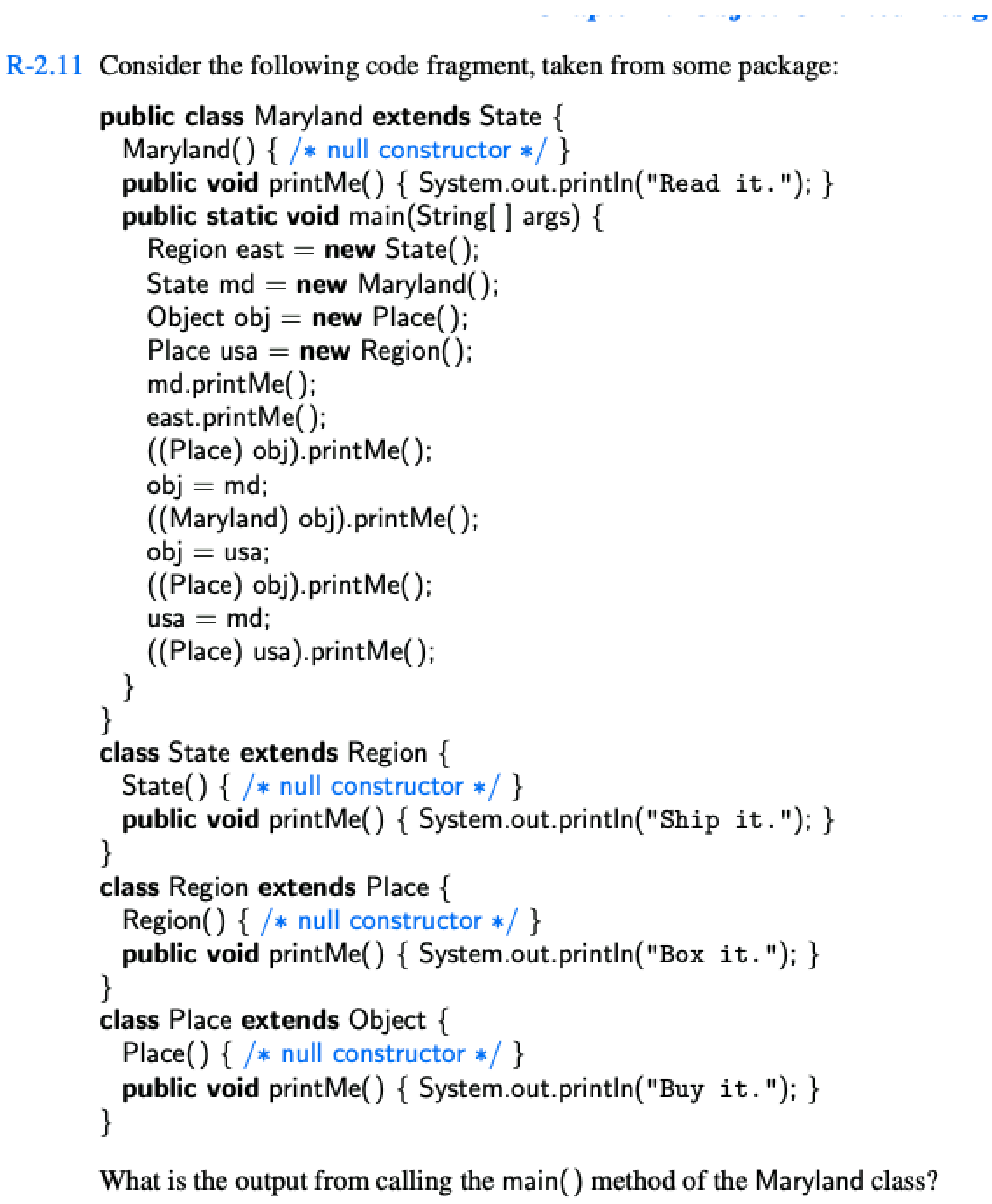
Two types:

Compile-time (Overloading)
Runtime(Overriding)



```
// Overloading
class MathUtil {
    int add(int a, int b) { return a + b; }
    double add(double a, double b) { return a + b; }
}

// Overriding
class Animal {
    void sound() { System.out.println("Animal sound"); }
}
class Dog extends Animal {
    void sound() { System.out.println("Bark"); }  // overrides
}
```

Let’s Do some Question -



R-2.11 Consider the following code fragment, taken from some package:

```
public class Maryland extends State {
  Maryland( ) { /* null constructor */ }
  public void printMe( ) { System.out.println("Read it."); }
  public static void main(String[ ] args) {
    Region east = new State( );
    State md = new Maryland( );
    Object obj = new Place( );
    Place usa = new Region( );
    md.printMe( );
    east.printMe( );
    ((Place) obj).printMe( );
    obj = md;
    ((Maryland) obj).printMe( );
    obj = usa;
    ((Place) obj).printMe( );
    usa = md;
    ((Place) usa).printMe( );
  }
}
class State extends Region {
  State( ) { /* null constructor */ }
  public void printMe( ) { System.out.println("Ship it."); }
}
class Region extends Place {
  Region( ) { /* null constructor */ }
  public void printMe( ) { System.out.println("Box it."); }
}
class Place extends Object {
  Place( ) { /* null constructor */ }
  public void printMe( ) { System.out.println("Buy it."); }
}
```

What is the output from calling the main( ) method of the Maryland class?

Object -> Place -> Region -> State -> Maryland

Outputs - md.printMe();

md is declared State but holds a Maryland object.

Dynamic dispatch picks Maryland’s override.

Output: **Read it.**

east.printMe();

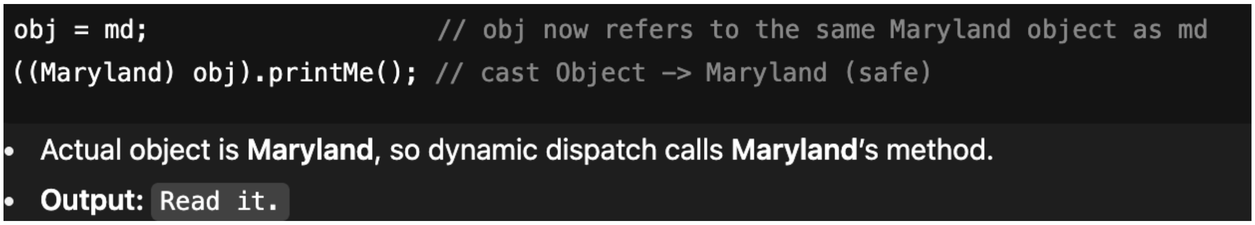
east is declared Region but holds a State object.

Dynamic dispatch picks State’s override.

Output: **Ship it.**

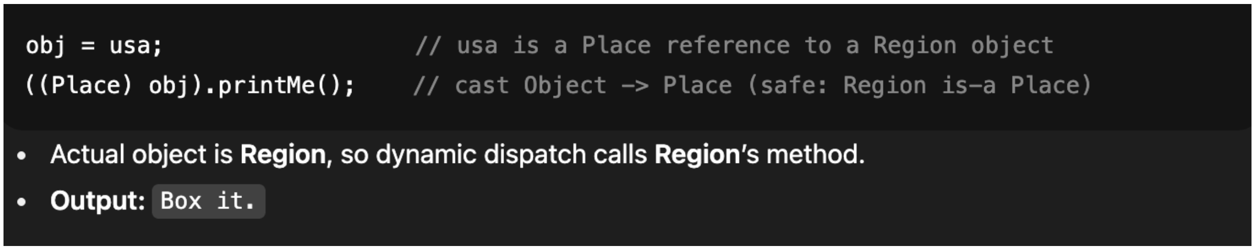
((Place) obj).printMe();

- obj currently refers to a Place object (declared Object).
- Casting to Place is safe because the actual object is Place.
- Call resolves to Place’s printMe().
- Output: **Buy it.**



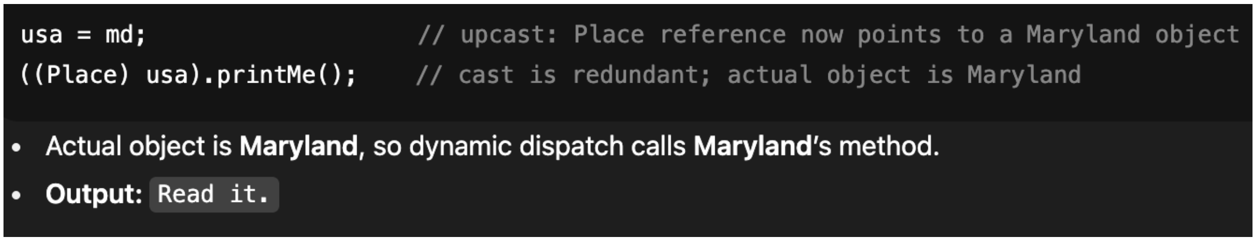
```
obj = md;                     // obj now refers to the same Maryland object as md
((Maryland) obj).printMe(); // cast Object -> Maryland (safe)
```

- Actual object is **Maryland**, so dynamic dispatch calls **Maryland**’s method.
- **Output:** `Read it.`



```
obj = usa;                  // usa is a Place reference to a Region object
((Place) obj).printMe();    // cast Object -> Place (safe: Region is-a Place)
```

- Actual object is **Region**, so dynamic dispatch calls **Region**’s method.
- **Output:** `Box it.`



```
usa = md;                   // upcast: Place reference now points to a Maryland object
((Place) usa).printMe();    // cast is redundant; actual object is Maryland
```

- Actual object is **Maryland**, so dynamic dispatch calls **Maryland**’s method.
- **Output:** `Read it.`

Final Order -

Read it.
Ship it.
Buy it.
Read it.
Box it.
Read it.

Questions to Try in a Group of 3 -

Anyone of these -

2.12, 2.13, 2.17