Part 1: Recall and Define

1. Method Overriding happens when a subclass **redefines** a method from its parent with the same **name** and **parameters**.
2. Method Overloading means creating methods with the same **name** but different **parameters**.
3. Dynamic Binding allows the program to decide which method to execute at **runtime**.
4. Upcasting means assigning a **subclass** object to a **superclass** reference.
5. Downcasting means converting a **superclass** reference back to a **subclass** type.

Part 2: Run the Code

(Output)

```
Java course
Programming course (Beginner)
Java-specific feature
Java course
Java course
Programming course (Advanced)
Java-specific feature
Programming course
```

Part 3: Think and Answer

رقم السؤال	الإجابة
1. Which methods are overridden?	`showInfo()` in `ProgrammingCourse` and `JavaCourse`.
2. Which method is overloaded?	`showInfo(String level)` in `ProgrammingCourse`.
3. What will be the output of the first method call `c1.showInfo()`?	`Java course` (because of dynamic binding — it runs the version in `JavaCourse`).
4. Why is the call `p1.showInfo("Advanced")` valid, even though the variable type is ProgrammingCourse?	Because the method `showInfo(String)` is **defined in `ProgrammingCourse`**, so the compiler allows it.

رقم السؤال	الإجابة
5. What is happening when this line executes: `JavaCourse j1 = (JavaCourse) c1;`	This is **downcasting**, converting the reference `c1` (of type `Course`) back to the subclass `JavaCourse` to access its unique methods.
6. What happens if `c1` refers to a ProgrammingCourse object and we still cast it to JavaCourse?	A **`ClassCastException`** will occur at runtime.
7. What Java concept ensures that `c1.showInfo()` calls the correct version at runtime?	**Dynamic Binding (Runtime Polymorphism)**.

Part 4: Practice Tasks

Task 1:

Answer:

```
class PythonCourse extends ProgrammingCourse {
    @Override
    public void showInfo() {
        System.out.println("Python course");
    }
}

// In main:
Course c = new Course();
ProgrammingCourse p = new ProgrammingCourse();
JavaCourse j = new JavaCourse();
PythonCourse py = new PythonCourse();

c.showInfo();
p.showInfo();
j.showInfo();
py.showInfo();
```

Output:

```
General course
Programming course
Java course
Python course
```

Task 2:

Question: Try to call `javaOnly()` using `Course c1 = new JavaCourse();` directly. What happens? Why? How can you fix it?

Answer:

- ❌ It causes a **compile-time error**:
  `cannot find symbol: method javaOnly()`
- **Reason:** The reference type `Course` doesn't know about the `javaOnly()` method (it's not declared in `Course`).
- **Fix:** Use **downcasting** to access it:
- `((JavaCourse) c1).javaOnly();`

Output after fix:

```
Java-specific feature
```

Challenge Section Example (Hospital Theme)

```
class Hospital {
    void showInfo() {
        System.out.println("General hospital");
    }
}

class Department extends Hospital {
    void showInfo() { // overriding
        System.out.println("Hospital department");
    }

    void showInfo(String dept) { // overloading
        System.out.println("Department: " + dept);
    }
}

class EmergencyDept extends Department {
    void showInfo() { // overriding again
        System.out.println("Emergency department");
    }

    void emergencyOnly() {
        System.out.println("Emergency team ready!");
    }
}

public class HospitalDemo {
    public static void main(String[] args) {
        Hospital h1 = new EmergencyDept();   // upcasting
        h1.showInfo();                       // dynamic binding
        Department d1 = new EmergencyDept();
        d1.showInfo("Cardiology");           // overloading
```

```
        EmergencyDept e1 = (EmergencyDept) h1; // downcasting
        e1.emergencyOnly();
    }
}
```

Reflection Answers

1. **Why is dynamic binding useful in real-world Java applications?**
   → It allows the correct method to run based on the object's actual type at runtime, enabling flexibility and extensibility (polymorphism).
2. **What are the risks of downcasting?**
   → If the object isn't really an instance of the subclass, it causes a **`ClassCastException`** at runtime.
3. **How does upcasting help achieve polymorphism?**
   → It allows one reference type (the superclass) to handle different subclass objects, letting us write generalized and reusable code.

Lab 6

(Polymorphism and dynamic binding)

```
// Parent class
class Course {
public void showInfo() {
System.out.println("General course");
}
}
// Child 1
class ProgrammingCourse extends Course {
// overriding
@Override
public void showInfo() {
```

```
System.out.println("Programming course");
}
// overloading (same name, different params)
public void showInfo(String level) {
System.out.println("Programming course (" + level + ")");
}
}
// Child 2 (more specific)
class JavaCourse extends ProgrammingCourse {
@Override
public void showInfo() {
System.out.println("Java course");
}
public void javaOnly() {
System.out.println("Java-specific feature");
}
}
```

```
class PythonCourse extends ProgrammingCourse {
    @Override
    public void showInfo() {
        System.out.println("Python course");
    }
}
```

```
public class CourseApp {
public static void main(String[] args) {
// 1) Normal object
JavaCourse j1 = new JavaCourse();
j1.showInfo(); // Java course
j1.showInfo("Beginner"); // from ProgrammingCourse (overloading)
j1.javaOnly();
// 2) Upcasting (actual = JavaCourse, declared = Course)
Course c1 = new JavaCourse(); // upcast
c1.showInfo(); // dynamic binding → calls JavaCourse.showInfo()
// 3) Another upcasting level
ProgrammingCourse p1 = new JavaCourse();
p1.showInfo(); // Java course (dynamic binding)
```

```
p1.showInfo("Advanced"); // calls overloaded version in ProgrammingCourse
// 4) Downcasting to access child-only method
if (c1 instanceof JavaCourse) {
JavaCourse j2 = (JavaCourse) c1; // downcast
j2.javaOnly();



Course c = new Course();
ProgrammingCourse p = new ProgrammingCourse();
JavaCourse j = new JavaCourse();
PythonCourse py = new PythonCourse();

c.showInfo();
p.showInfo();
j.showInfo();
py.showInfo();

}
// 5) Another example of overriding
Course c2 = new ProgrammingCourse();
c2.showInfo(); // Programming course (not General course)
}
                                                                                }
```