Java 21

Pattern Matching for ‘switch’

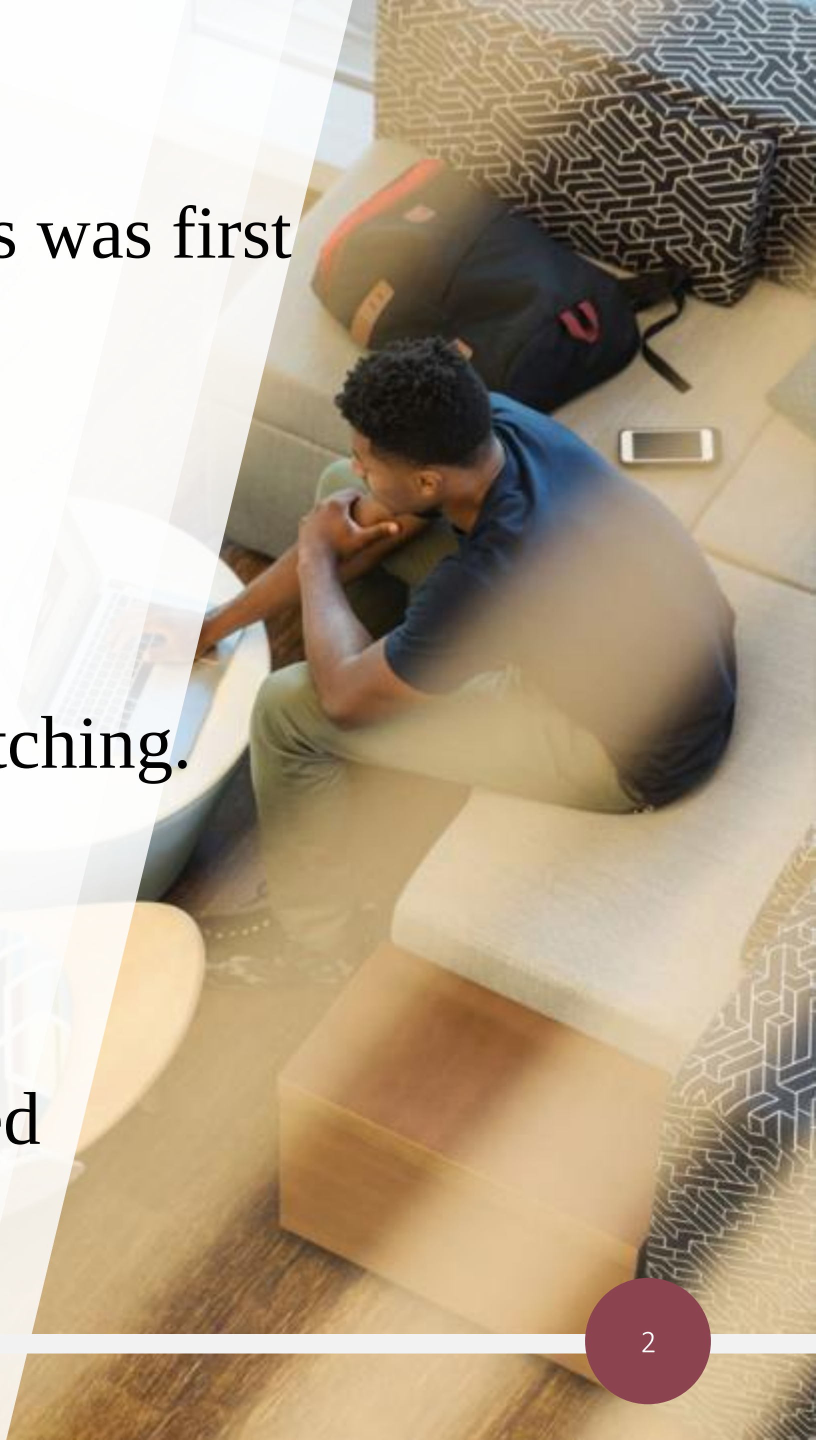
Pattern-matching for *switch*

- Pattern matching for *switch* statements and expressions was first introduced as a preview feature in Java 17.

- Now, Java 21 finalizes the feature.

- Motivation – *switch* is a very natural fit for pattern matching. Recall that pattern matching removes the need for the *instanceof*-and-cast idiom.

- Other changes, such as the *when* clause, were motivated by the desire to separate the *case* labels, patterns and conditional logic from the business logic.

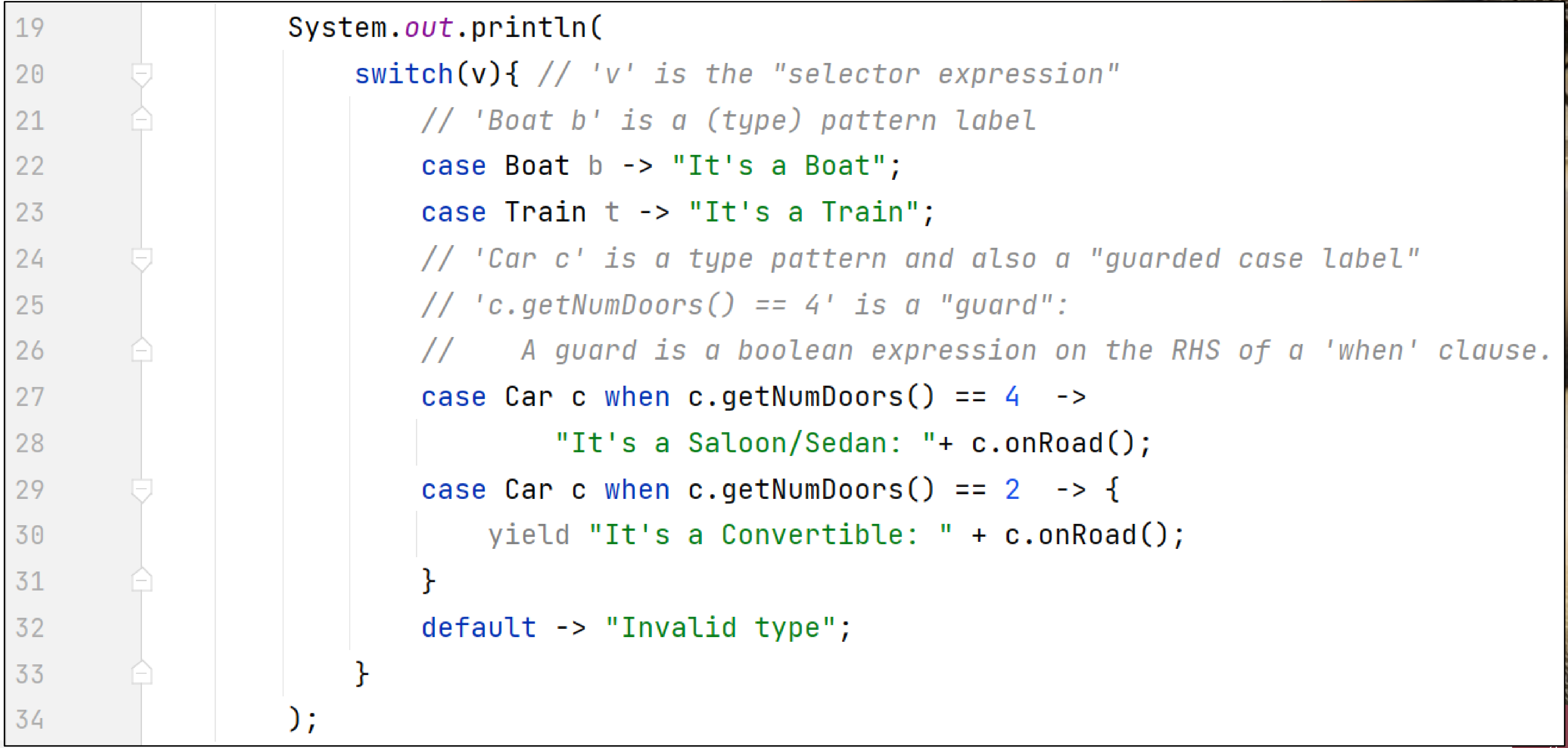


Pattern-matching for *switch*

- In other words, the selection of which branch to execute is separated from what to do when we execute that branch.

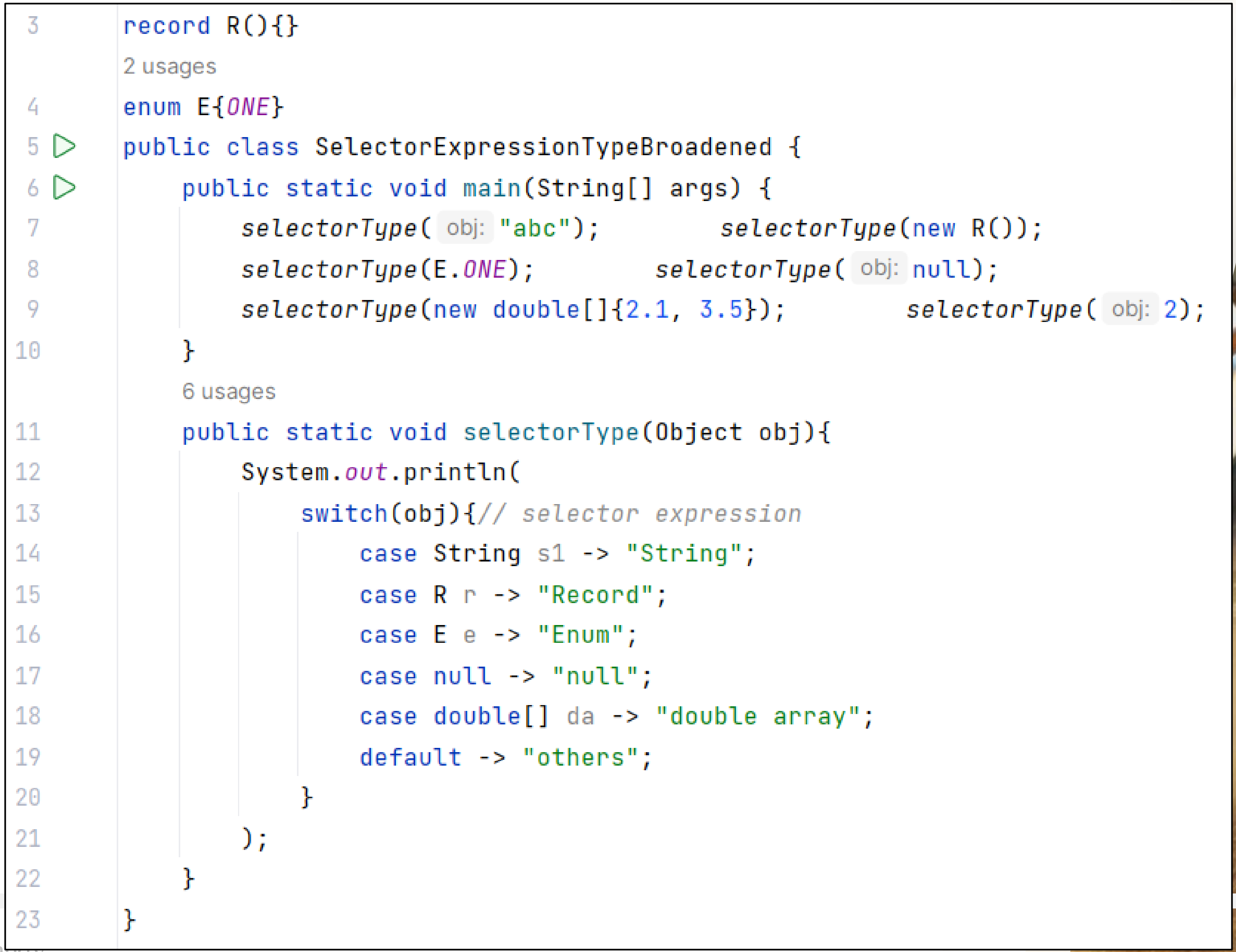
- Changes include:
  - *case* labels can include patterns and *null*
  - *case* labels can include optional *when* clauses (“guards”)
  - selector expression types broadened
    - from:
      - integral primitive types (excluding *long*), their corresponding wrapper types, *String* and enums.
    - to:
      - integral primitive types (excluding *long*) and any reference type.
  - *enum* constant *case* labels improved
    - qualified *enum* constants now allowed

Pattern-matching for *switch* - pattern labels, *null* and *when* clauses



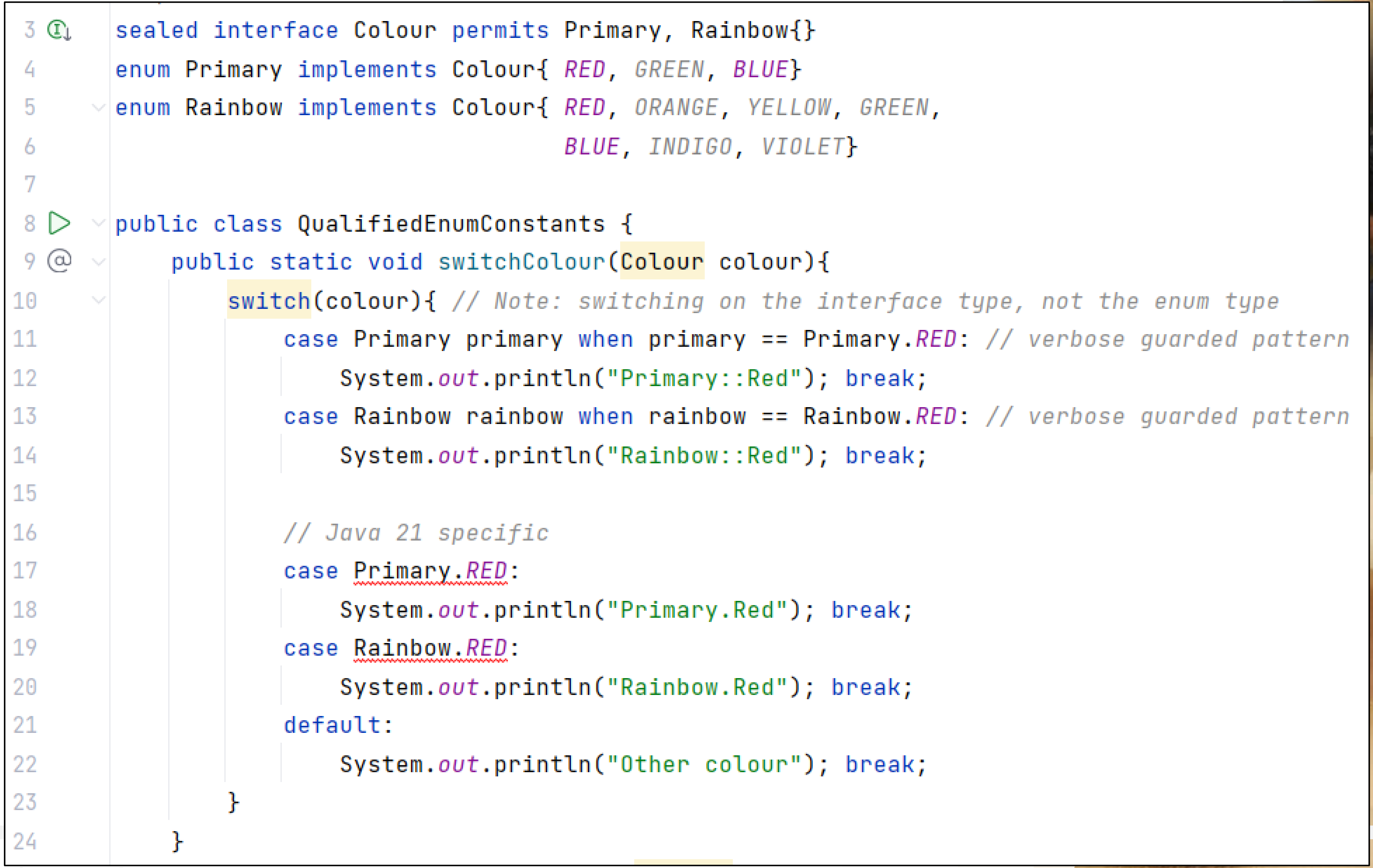
```
        System.out.println(
            switch(v){ // 'v' is the "selector expression"
                // 'Boat b' is a (type) pattern label
                case Boat b -> "It's a Boat";
                case Train t -> "It's a Train";
                // 'Car c' is a type pattern and also a "guarded case label"
                // 'c.getNumDoors() == 4' is a "guard":
                //     A guard is a boolean expression on the RHS of a 'when' clause.
                case Car c when c.getNumDoors() == 4  ->
                        "It's a Saloon/Sedan: "+ c.onRoad();
                case Car c when c.getNumDoors() == 2  -> {
                    yield "It's a Convertible: " + c.onRoad();
                }
                default -> "Invalid type";
            }
        );
```

Pattern-matching for *switch* - selector expression types broadened



```
record R(){}
2 usages
enum E{ONE}
public class SelectorExpressionTypeBroadened {
    public static void main(String[] args) {
        selectorType( obj: "abc");        selectorType(new R());
        selectorType(E.ONE);        selectorType( obj: null);
        selectorType(new double[]{2.1, 3.5});        selectorType( obj: 2);
    }
    6 usages
    public static void selectorType(Object obj){
        System.out.println(
            switch(obj){// selector expression
                case String s1 -> "String";
                case R r -> "Record";
                case E e -> "Enum";
                case null -> "null";
                case double[] da -> "double array";
                default -> "others";
            }
        );
    }
}
```

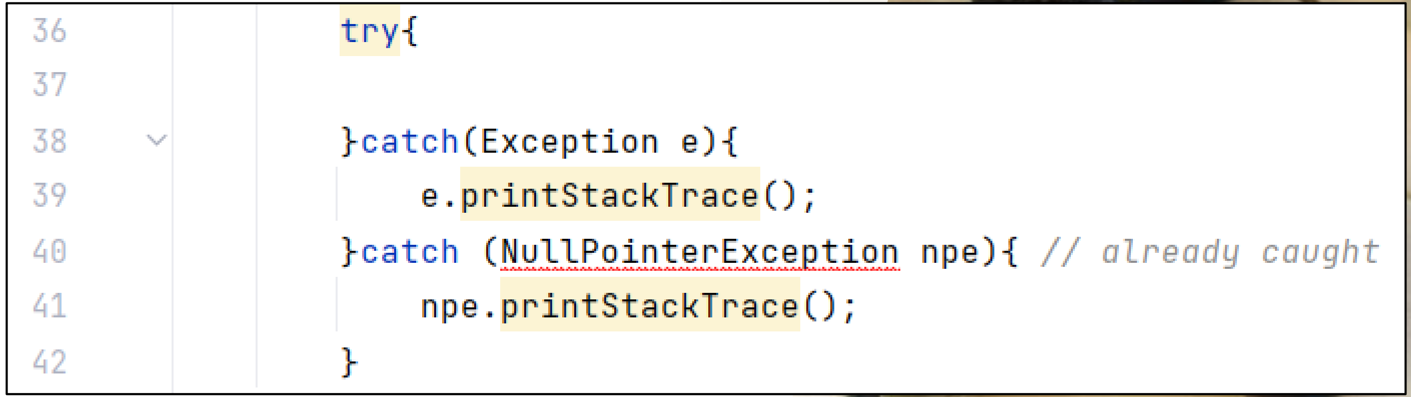
Pattern-matching for *switch* – *enum* constant *case* labels improved



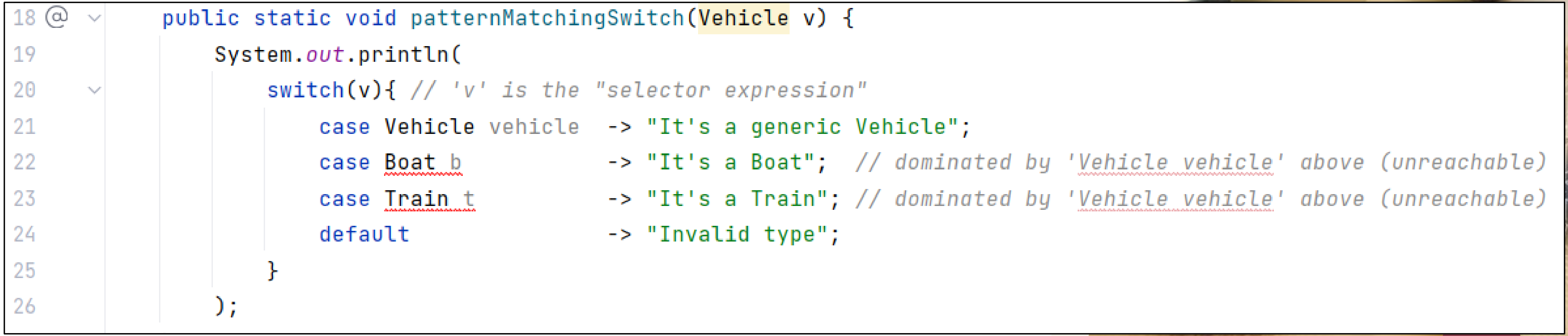
```java
sealed interface Colour permits Primary, Rainbow{}
enum Primary implements Colour{ RED, GREEN, BLUE}
enum Rainbow implements Colour{ RED, ORANGE, YELLOW, GREEN,
                                BLUE, INDIGO, VIOLET}

public class QualifiedEnumConstants {
    public static void switchColour(Colour colour){
        switch(colour){ // Note: switching on the interface type, not the enum type
            case Primary primary when primary == Primary.RED: // verbose guarded pattern
                System.out.println("Primary::Red"); break;
            case Rainbow rainbow when rainbow == Rainbow.RED: // verbose guarded pattern
                System.out.println("Rainbow::Red"); break;

            // Java 21 specific
            case Primary.RED:
                System.out.println("Primary.Red"); break;
            case Rainbow.RED:
                System.out.println("Rainbow.Red"); break;
            default:
                System.out.println("Other colour"); break;
        }
    }
```

Pattern-matching for *switch*

- Label “dominance”
  - analogous to the *catch* clauses in a *try* statement
    - unreachable code (label)



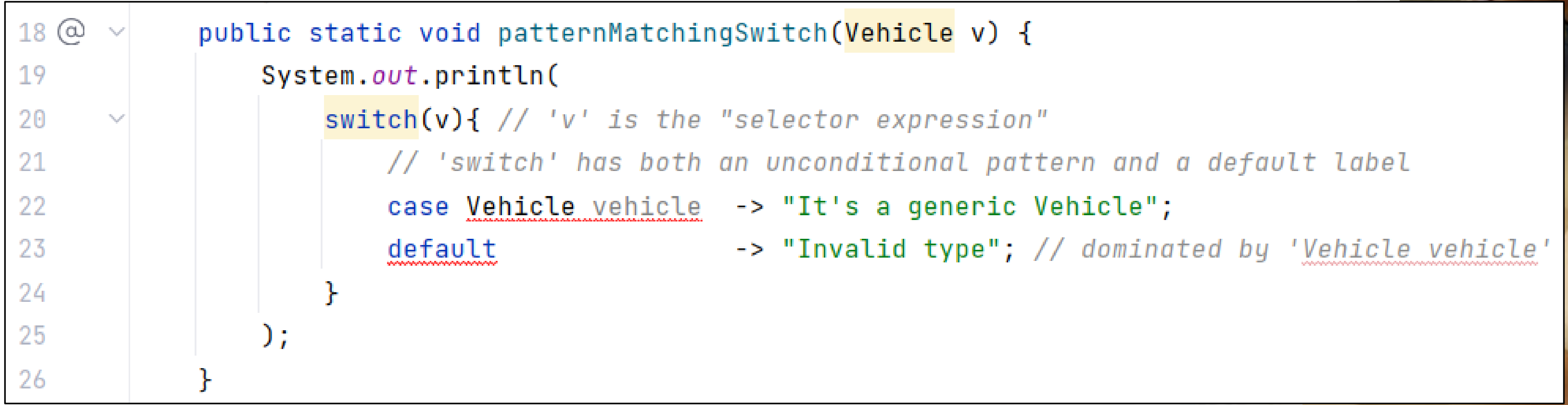
```
        try{

        }catch(Exception e){
            e.printStackTrace();
        }catch (NullPointerException npe){ // already caught
            npe.printStackTrace();
        }
```



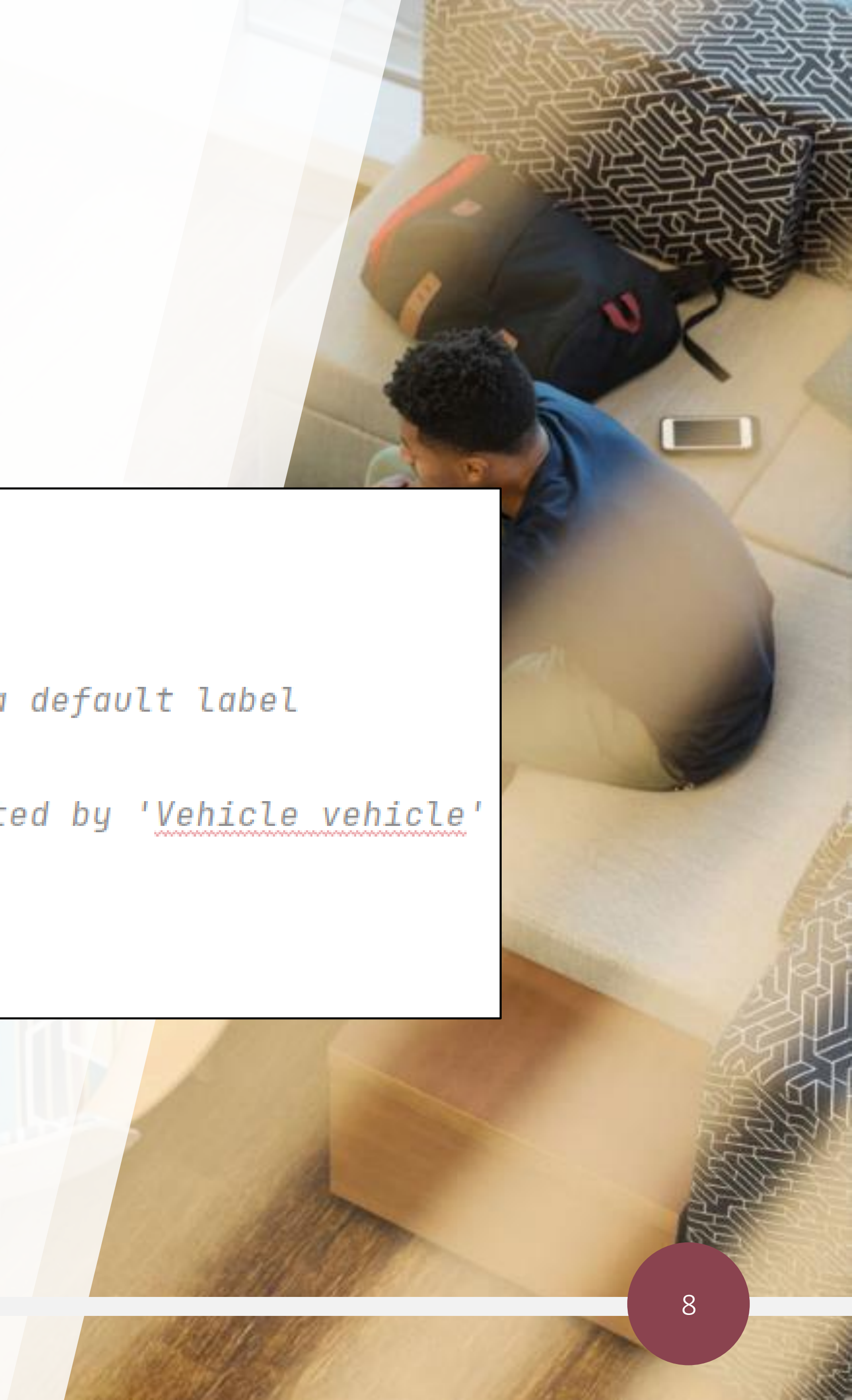
```
    public static void patternMatchingSwitch(Vehicle v) {
        System.out.println(
            switch(v){ // 'v' is the "selector expression"
                case Vehicle vehicle  -> "It's a generic Vehicle";
                case Boat b           -> "It's a Boat";  // dominated by 'Vehicle vehicle' above (unreachable)
                case Train t          -> "It's a Train"; // dominated by 'Vehicle vehicle' above (unreachable)
                default               -> "Invalid type";
            }
        );
```

Pattern-matching for *switch*

- Label "dominance"
  - unconditional pattern and *default*

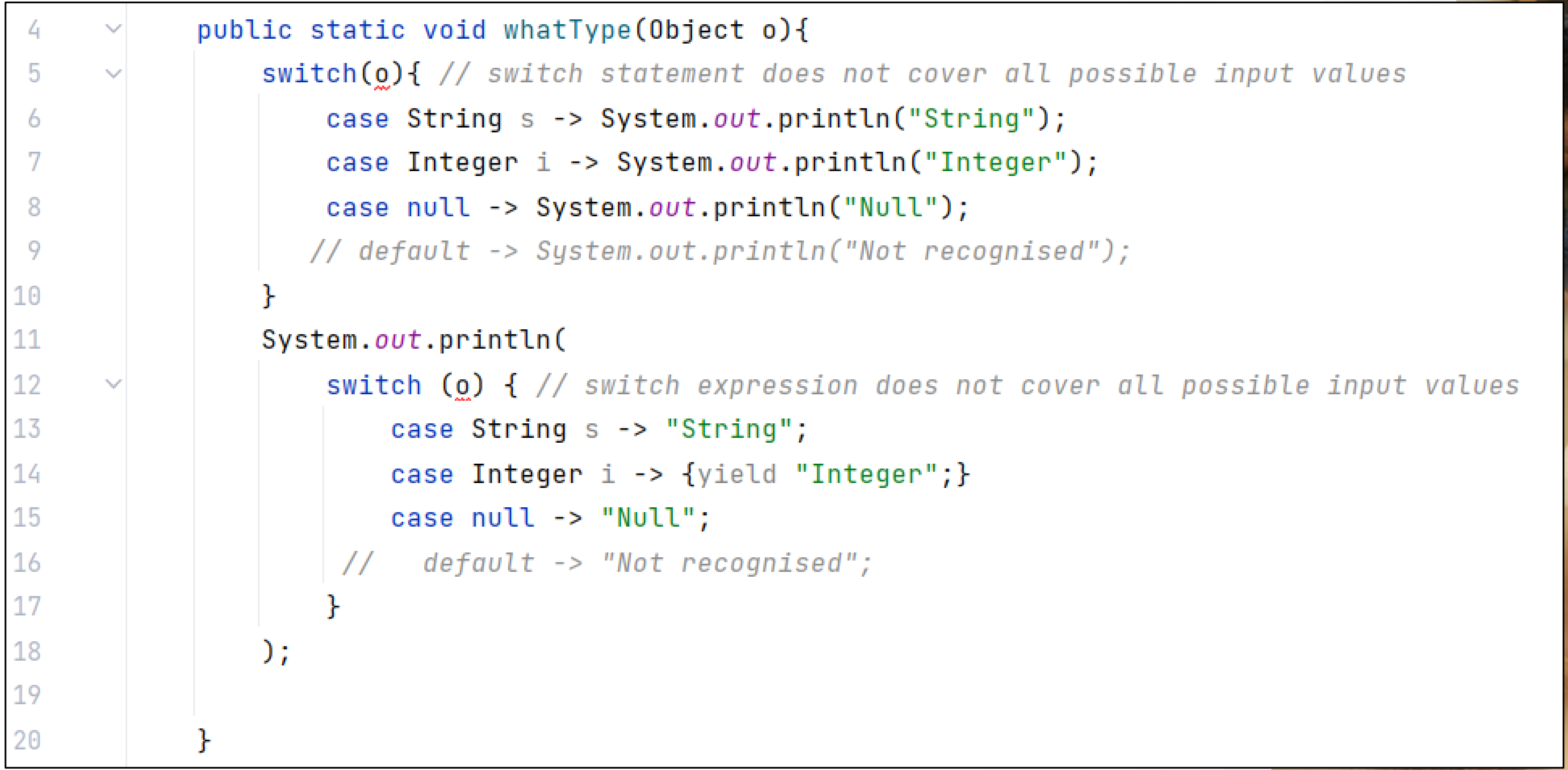


```
    public static void patternMatchingSwitch(Vehicle v) {
        System.out.println(
            switch(v){ // 'v' is the "selector expression"
                // 'switch' has both an unconditional pattern and a default label
                case Vehicle vehicle  -> "It's a generic Vehicle";
                default               -> "Invalid type"; // dominated by 'Vehicle vehicle'
            }
        );
    }
```



Pattern-matching for *switch*

- Exhaustiveness



```java
    public static void whatType(Object o){
        switch(o){ // switch statement does not cover all possible input values
            case String s -> System.out.println("String");
            case Integer i -> System.out.println("Integer");
            case null -> System.out.println("Null");
           // default -> System.out.println("Not recognised");
        }
        System.out.println(
            switch (o) { // switch expression does not cover all possible input values
                case String s -> "String";
                case Integer i -> {yield "Integer";}
                case null -> "Null";
            //   default -> "Not recognised";
            }
        );

    }
```