12

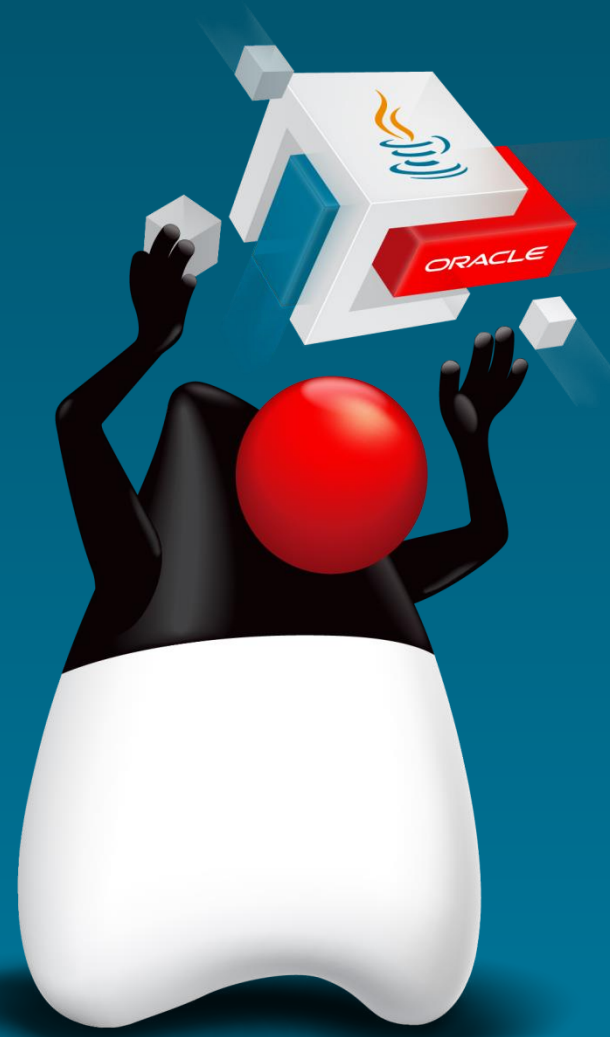# Using Inheritance

Interactive Quizzes

Objectives

After completing this lesson, you should be able to:

- Define inheritance in the context of a Java class hierarchy

- Create a subclass

- Override a method in the superclass

- Use the `super` keyword to reference the superclass

- Define polymorphism

- Use the `instanceof` operator to test an object's type

- Cast a superclass reference to the subclass type

- Explain the difference between abstract and non-abstract classes

- Create a class hierarchy by extending an abstract class

Duke's Choice Classes: Common Behaviors

Shirt	Trousers
getId() **getPrice()** **getSize()** **getColor()** **getFit()**	getId() getPrice() getSize() getColor() getFit() getGender()
setId() **setPrice()** **setSize()** **setColor()** **setFit()**	setId() setPrice() setSize() setColor() setFit() setGender()
display()	display()

Code Duplication

Shirt
getId()
display()
getPrice()
getSize()
getColor()
getFit()

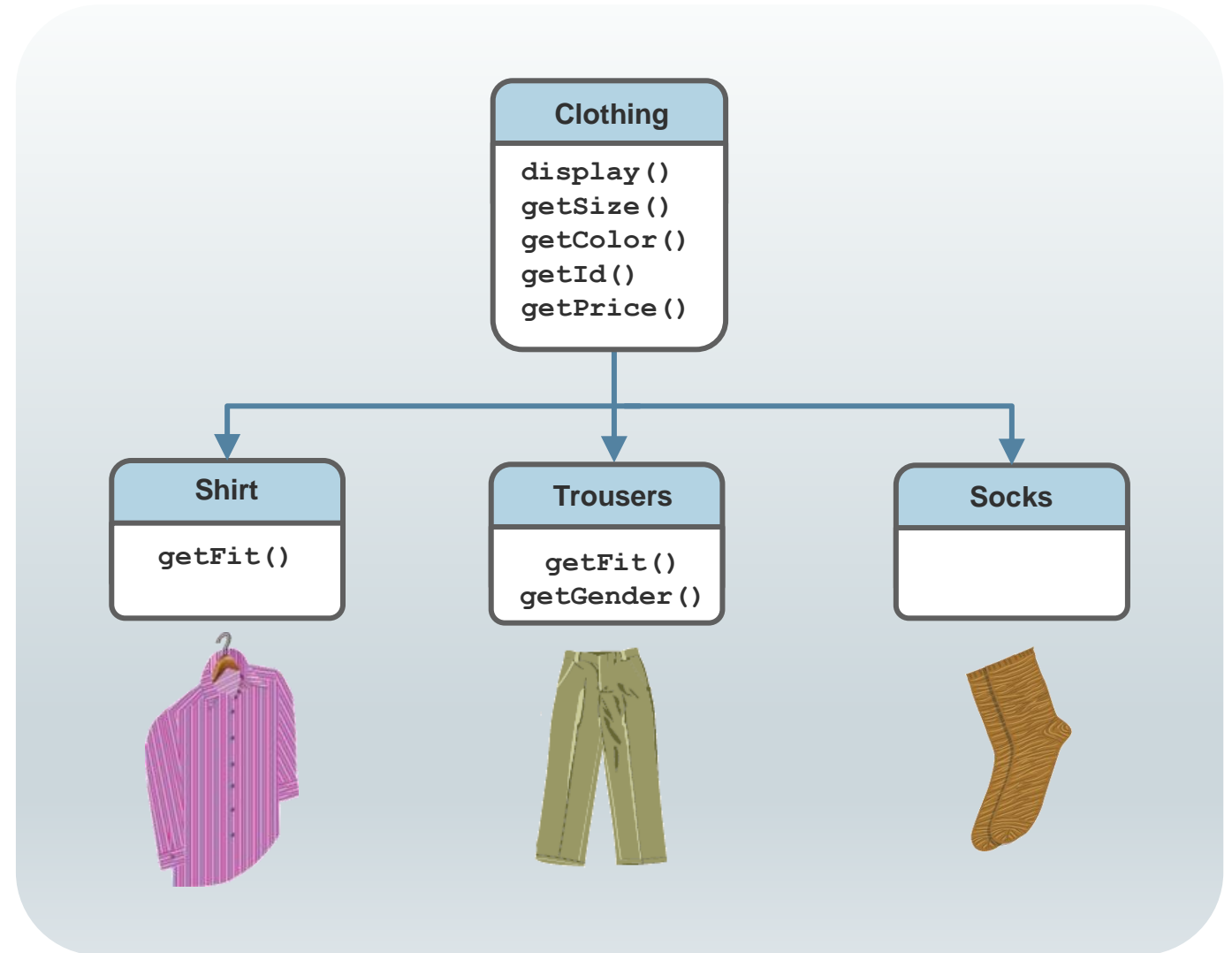| Trousers |
|---|
| getId() |
| display() |
| getPrice() |
| getSize() |
| getColor() |
| getFit() |
| getGender() |

Socks
getId()
display()
getPrice()
getSize()
getColor()

Inheritance

- **Inheritance** allows one class to be derived from another.

- Fields and methods are written in one class, and then inherited by other classes.

    – There is less code duplication.

    – Edits are done in one location

Inheritance Terminology

- The term inheritance is inspired by biology
  - A child inherits properties and behaviors of the parent.
  - A child *class* inherits the fields and method of a parent *class.*

- The parent class is known as the **superclass**.
  - A superclass is the common location for fields and methods.

- The child class is known as the **subclass**. A subclass extends its superclass.
  - Subclasses share the same methods as the superclass.
  - Subclasses may have additional methods that aren't found in their superclass.
  - Subclasses may override the methods they inherit from their superclass.

Topics

- Overview of inheritance
- **Working with superclasses and subclasses**
- Overriding superclass methods
- Introducing polymorphism
- Creating and extending abstract classes

Implementing Inheritance

```java
public class Clothing {
    public void display(){…}
    public void setSize(char size){…}
}
```

```java
public class Shirt extends Clothing {…}
```

Use the extends keyword.

```java
Shirt myShirt = new Shirt();
myShirt.setSize ('M');
```

This code works!

More Inheritance Facts

- A subclass has access to all of the public fields and methods of its superclass.
- A subclass may have unique fields and methods not found in the superclass.

subclass   superclass

```java
public class Shirt extends Clothing {
    private int neckSize;
    public int getNeckSize(){
        return neckSize;
    }
    public void setNeckSize(int nSize){
        this.neckSize = nSize;
    }
}
```

`Clothing` Class: Part 1

```
01 public class Clothing {
02    // fields given default values
03    private int itemID = 0;
04    private String desc = "-description required-";
05    private char colorCode = 'U';
06    private double price = 0.0;
07
08    // Constructor
09    public Clothing(int itemID, String desc, char color,
10       double price ) {
11        this.itemID = itemID;
12        this.desc = desc;
13        this.colorCode = color;
14        this.price = price;
15    }
16 }
```

`Shirt` Class: Part 1

```
01  public class Shirt extends Clothing {
03     private char fit = 'U';
04
05     public Shirt(int itemID, String description, char
06              colorCode, double price, char fit) {
07         super (itemID, description, colorCode, price);
08
09         this.fit = fit;
10     }
12     public char getFit() {
13         return fit;
14     }
15     public void setFit(char fit) {
16         this.fit = fit;
17     }
```

Reference to the superclass constructor

Reference to this object

Constructor Calls with Inheritance

```java
public static void main(String[] args){
    Shirt shirt01 = new Shirt(20.00, 'M');    }
```

```java
public class Shirt extends Clothing {
    private char fit = 'U';

    public Shirt(double price, char fit) {
        super(price);          //MUST call superclass constructor
        this.fit = fit;        }}
```

```java
public class Clothing{
    private double price;

    public Clothing(double price){
        this.price = price;
}}
```

Inheritance and Overloaded Constructors

```java
public class Shirt extends Clothing {
  private char fit = 'U';

  public Shirt(char fit){
     this(15.00, fit);      //Call constructor in same class
  }                         //Constructor is overloaded


  public Shirt(double price, char fit) {
     super(price);          //MUST call superclass constructor
     this.fit = fit;        }}
```

```java
  public class Clothing{
    private double price;


    public Clothing(double price){
        this.price = price;
}}
```

Exercise 12-1: Creating a Subclass, Part 1

1. Open the project **Exercise_12-1**.

2. Examine the `Item` class. Pay close attention to the overloaded constructor and also the `display` method.

3. In the **exercise_12_1** package, create a new class called `Shirt` that inherits from Item.

4. In the `Shirt` class, declare two private `char` fields: `size` and `colorCode`.

5. Create a constructor method that takes 3 args (`price`, `size`, `colorCode`). The constructor should:
   - Call the 2-arg constructor in the superclass
     - Pass a String literal for the `desc` arg ("Shirt").
     - Pass the price argument from this constructor.
   - Assign the size and colorCode fields.

Exercise 12-1: Creating a Subclass, Part 2

In the `ShoppingCart` class:

6. Declare and instantiate a `Shirt` object, using the 3-arg constructor.
7. Call the `display` method on the object reference.
   - Notice where the `display` method is actually coded.

Topics

- Overview of inheritance
- Working with superclasses and subclasses
- **Overriding superclass methods**
- Introducing polymorphism
- Creating and extending abstract classes

More on Access Control

Access level modifiers determine whether other classes can use a particular field or invoke a particular method

- At the top level—public, or *package-private* (no explicit modifier).

- At the member level—public, private, protected, or *package-private* (no explicit modifier).

Stronger
access privileges

| Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
No modifier	Y	Y	N	N
private	Y	N	N	N

Overriding Methods

Overriding: A subclass implements a method that already has an implementation in the superclass.

Access Modifiers:

- The method can only be overridden if it is accessible from the subclass

- The method signature in the subclass cannot have a more restrictive (stronger) access modifier than the one in the superclass

Review: Duke's Choice Class Hierarchy

Now consider these classes in more detail.
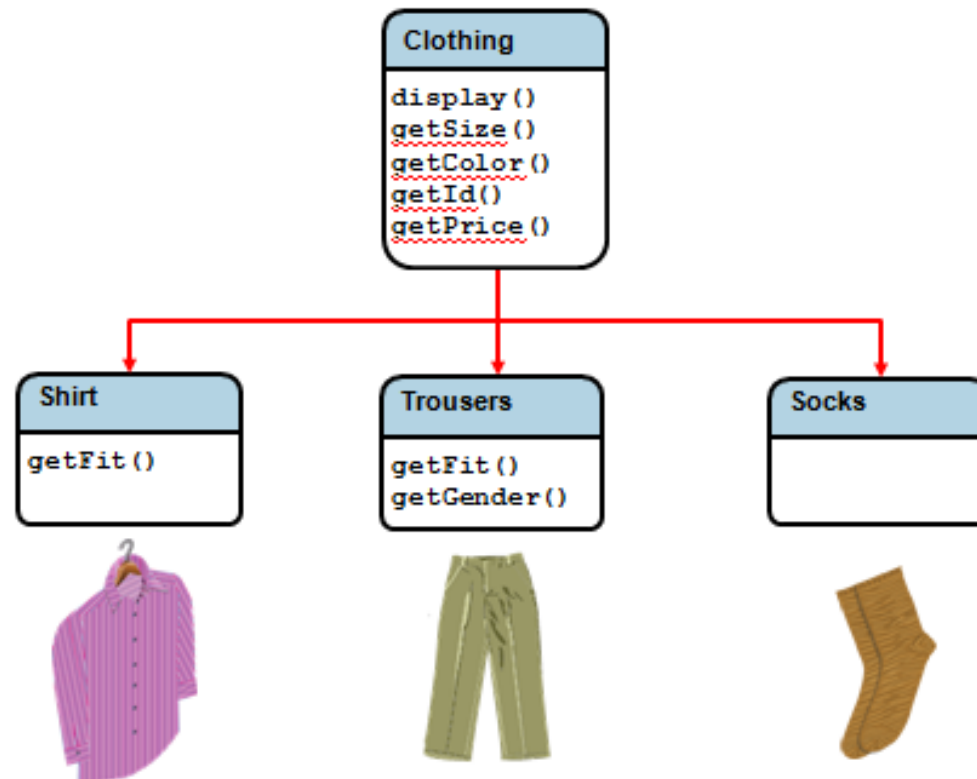
Clothing Class: Part 2

```java
29   public void display() {
30     System.out.println("Item ID: " + getItemID());
31     System.out.println("Item description: " + getDesc());
32     System.out.println("Item price: " + getPrice());
33     System.out.println("Color code: " + getColorCode());
34   }
35   public String getDesc (){
36       return desc;
37   }
38   public double getPrice() {
39       return price;
40   }
41   public int getItemID() {
42       return itemID;
43   }
44   protected void setColorCode(char color){
45       this.colorCode = color; }
```

Assume that the remaining `get/set` methods are included in the class.

`Shirt` Class: Part 2

```java
17  // These methods override the methods in Clothing
18  public void display() {
19      System.out.println("Shirt ID: " + getItemID());
20      System.out.println("Shirt description: " + getDesc());
21      System.out.println("Shirt price: " + getPrice());
22      System.out.println("Color code: " + getColorCode());
23      System.out.println("Fit: " + getFit());
24  }
25
26  protected void setColorCode(char colorCode) {
27      //Code here to check that correct codes used
28      super.setColorCode(colorCode);
29  }
30}
```

Call the superclass's version of `setColorCode`.

Overriding a Method: What Happens at Run Time?
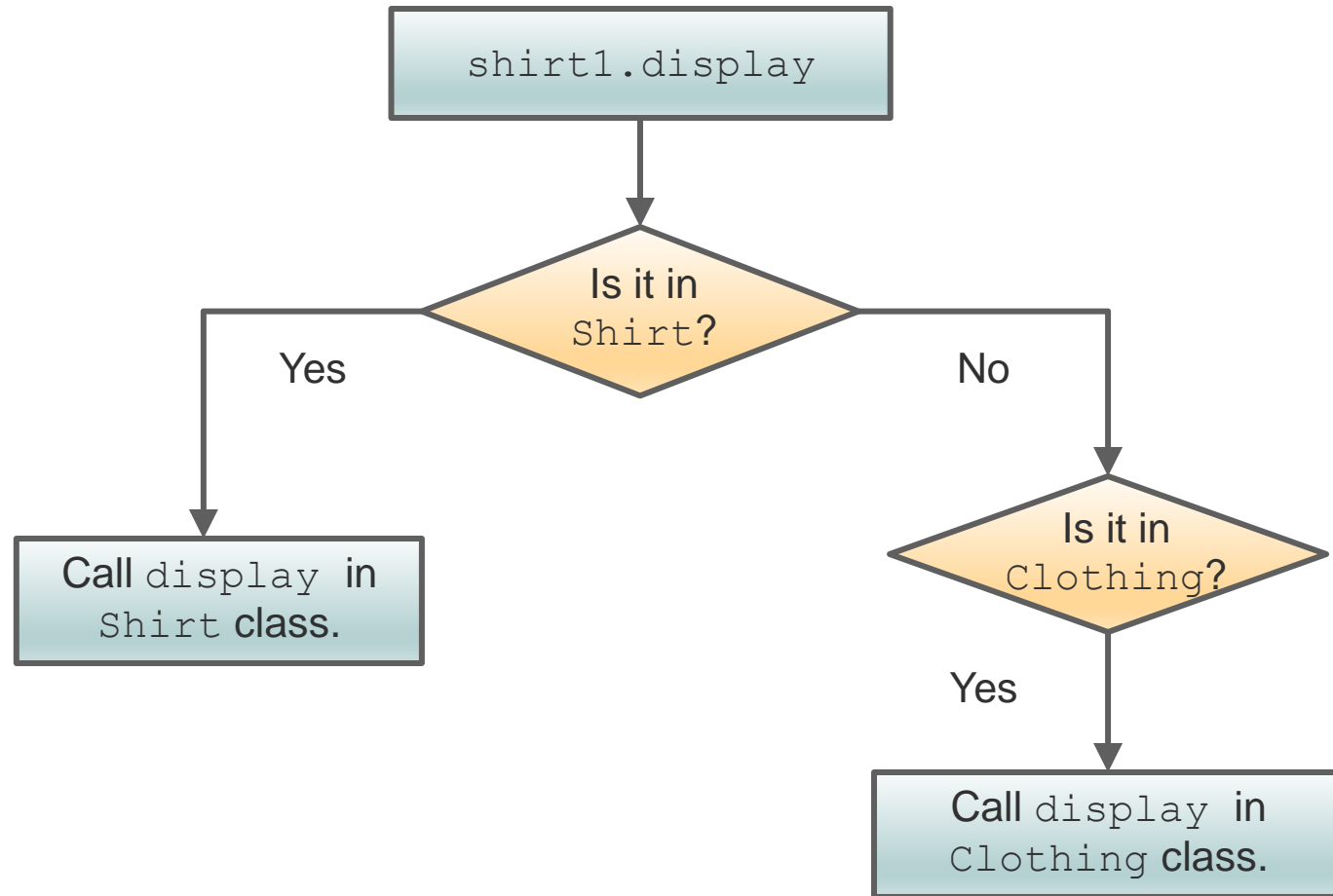
Exercise 12-2: Overriding a Method in the Superclass

1. Open **Exercise_12-2** or continue editing **Exercise_12-1**.

In the `Shirt` class:

2. Override the `display` method to do the following:
   - Call the superclass's display method.
   - Print the `size` field and the `colorCode` field.

3. Run the code. Do you see a different display than you did in the previous exercise?

Topics

- Overview of inheritance
- Working with superclasses and subclasses
- Overriding superclass methods
- **Introducing polymorphism**
- Creating and extending abstract classes

Polymorphism

- Polymorphism means that the same message to two different objects can have different results.

  - "Good night" to a child means "Start getting ready for bed."

  - "Good night" to a parent means "Read a bedtime story."

- In Java, it means the same method is implemented differently by different classes.

  - This is especially powerful in the context of inheritance.

  - It relies upon the "is a" relationship.

Superclass and Subclass Relationships

Use inheritance only when it is completely valid or unavoidable.

- Use the "*is a*" test to decide whether an inheritance relationship makes sense.

- Which of the phrases below expresses a valid inheritance relationship within the Duke's Choice hierarchy?

  OK
  - A Shirt *is a* piece of Clothing.
  - A Hat *is a* Sock.
  - Equipment *is a* piece of Clothing.

  OK
  - Clothing and Equipment *are* Items.

Using the Superclass as a Reference

So far, you have referenced objects only with a reference variable of the same class:

- To use the `Shirt` class as the reference type for the `Shirt` object:

```
Shirt myShirt = new Shirt();
```

- But you can also use the superclass as the reference:

```
Clothing garment1 = new Shirt();
Clothing garment2 = new Trousers();
```

Shirt is a (type of) Clothing.
Trousers is a (type of) Clothing.

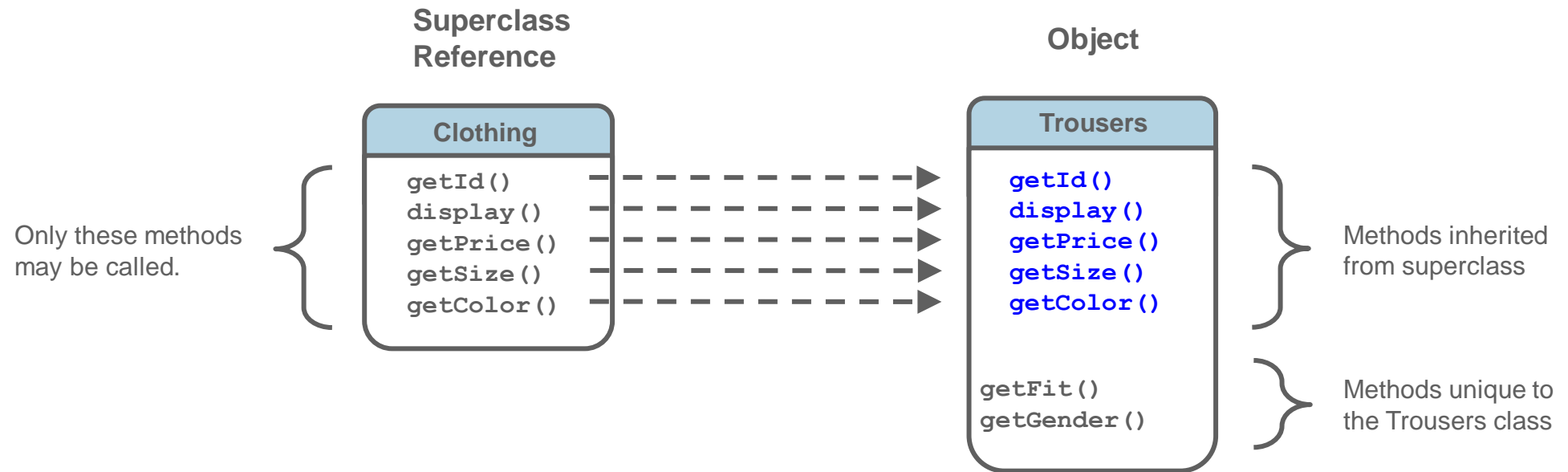# Polymorphism Applied

```
Clothing c1 = new ??();


c1.display();

c1.setColorCode('P');
```

c1 could be a Shirt, Trousers, or Socks object.

The method will be implemented differently on different types of objects. For example:
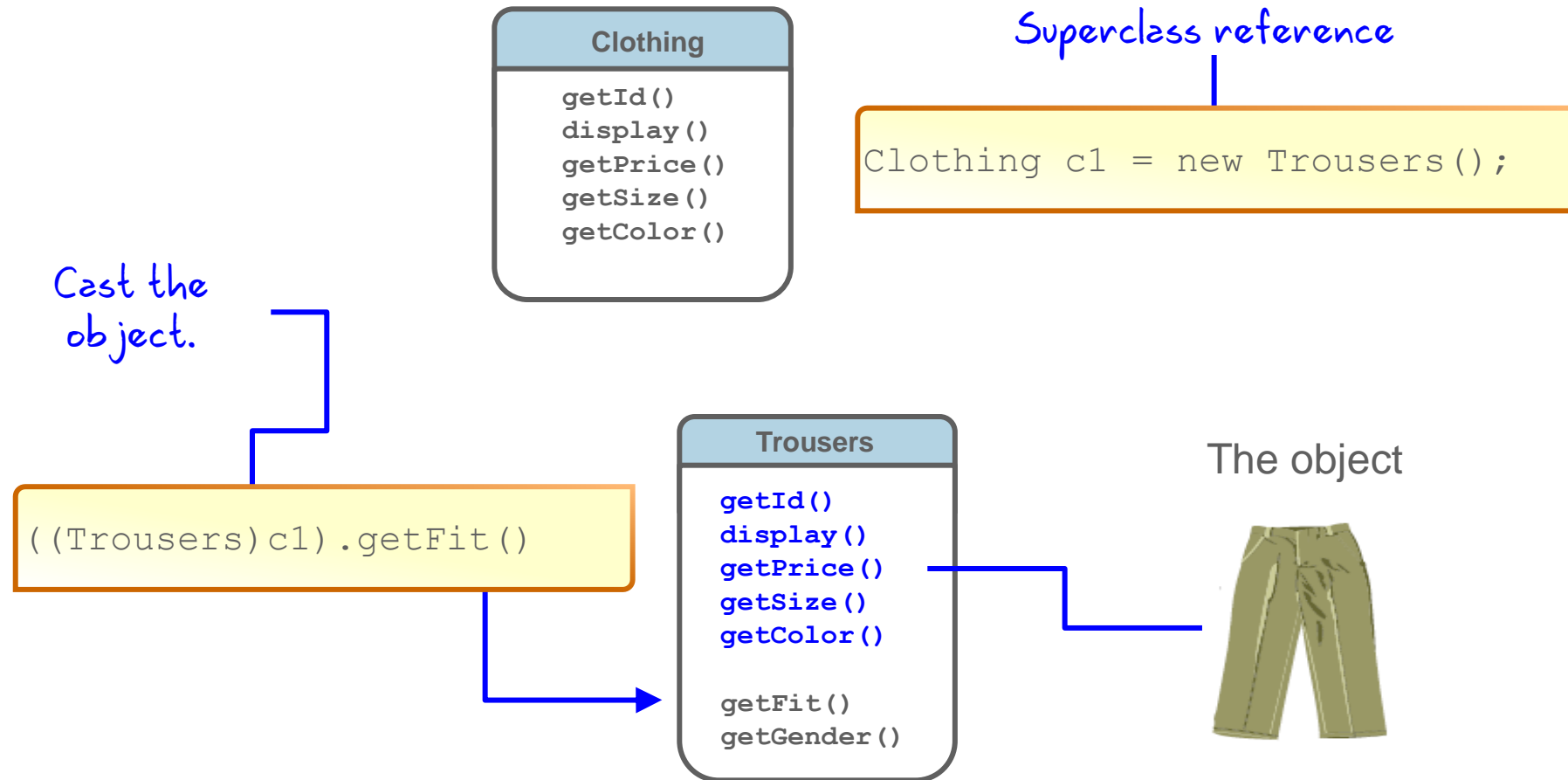
- Trousers objects show more fields in the display method.
- Different subclasses accept a different subset of valid color codes.

Accessing Methods Using a Superclass Reference

Superclass Reference

Object

Clothing

getId()
display()
getPrice()
getSize()
getColor()

Only these methods may be called.

Trousers

getId()
display()
getPrice()
getSize()
getColor()

Methods inherited from superclass

getFit()
getGender()

Methods unique to the Trousers class

```
Clothing c1 = new Trousers();
c1.getId();   OK
c1.display(); OK
c1.getFit();  NO!
```

Casting the Reference Type

Clothing

```
getId()
display()
getPrice()
getSize()
getColor()
```

Superclass reference

```
Clothing c1 = new Trousers();
```

Cast the object.

```
((Trousers)c1).getFit()
```

Trousers

```
getId()
display()
getPrice()
getSize()
getColor()

getFit()
getGender()
```

The object

`instanceof` Operator

Possible casting error:

```java
public static void displayDetails(Clothing cl) {

    cl.display();
    char fitCode = ((Trousers)cl).getFit();
    System.out.println("Fit: " + fitCode);

}
```

What if c1 is not a Trousers object?

`instanceof` operator used to ensure there is no casting error:

```java
public static void displayDetails(Clothing cl) {
    cl.display();
    if (cl instanceof Trousers) {
        char fitCode = ((Trousers)cl).getFit();
        System.out.println("Fit: " + fitCode);
    }
    else { // Take some other action }
```

`instanceOf` returns true if c1 is a Trousers object.

Exercise 12-3: Using the `instanceof` Operator, Part 1

1. Open **Exercise_12-3** or continue editing **Exercise_12-2**.

In the `Shirt` class:

2. Add a public `getColor` method that converts the `colorCode` field into the corresponding color name:
   - Example: 'R' = "Red"
   - Include at least 3 colorCode/color combinations.
3. Use a `switch` statement in the method and return the color String.

Exercise 12-3: Using the `instanceof` Operator, Part 2

In the `ShoppingCart` class:

4. Modify the `Shirt` object's declaration so that it uses an `Item` reference type instead.

5. Call the `display` method of the object.

6. Use `instanceof` to confirm that the object is a `Shirt`.
   - If it is a `Shirt`:
     - Cast the object to a `Shirt` and call the `getColor` method, assigning the return value to a String variable.
     - Print out the color name using a suitable label.
   - If it is not a `Shirt`, print a message to that effect.

7. Test your code. You can test the non-Shirt object condition by instantiating an Item object instead of a `Shirt` object.

Topics

- Overview of inheritance
- Working with superclasses and subclasses
- Overriding superclass methods
- Introducing polymorphism
- **Creating and extending abstract classes**
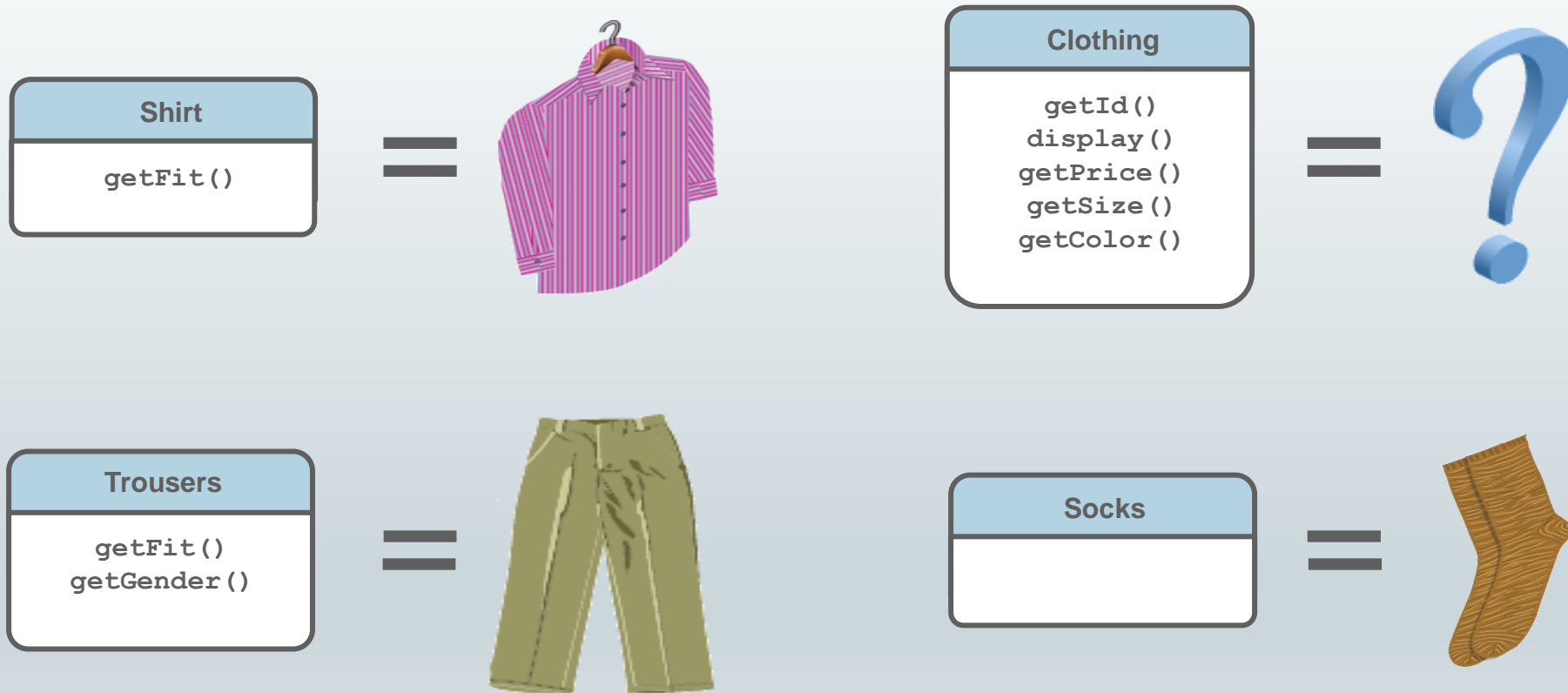
Abstract Classes

Abstract Classes

Use the `abstract` keyword to create a special class that:

- Cannot be instantiated
- May contain concrete methods
- May contain abstract methods that **must** be implemented later by any non-abstract subclasses

🚫 `Clothing cloth01 = new Clothing()`

```
public abstract class Clothing{
    private int id;

     public int getId(){                      Concrete
        return id;                             method
     }

    public abstract double getPrice();         Abstract
    public abstract void display();            methods

}
```

Extending Abstract Classes

```java
public abstract class Clothing{
    private int id;

    public int getId(){
        return id;
    }
    protected abstract double getPrice();    //MUST be implemented
    public abstract void display();   }      //MUST be implemented
```

```java
public class Socks extends Clothing{
    private double price;

    protected double getPrice(){
        return price;
    }
    public void display(){
        System.out.println("ID: " +getID());
        System.out.println("Price: $" +getPrice());
}}
```

Summary

In this lesson, you should have learned the following:

- Define inheritance in the context of a Java class hierarchy

- Create a subclass

- Override a method in the superclass

- Use the `super` keyword to reference the superclass

- Define polymorphism

- Use the `instanceof` operator to test an object's type

- Cast a superclass reference to the subclass type

- Explain the difference between abstract and non-abstract classes

- Create a class hierarchy by extending an abstract class

Practice Overview

- 12-1: Creating a Class Hierarchy
- 12-2: Creating a `GameEvent` Hierarchy