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

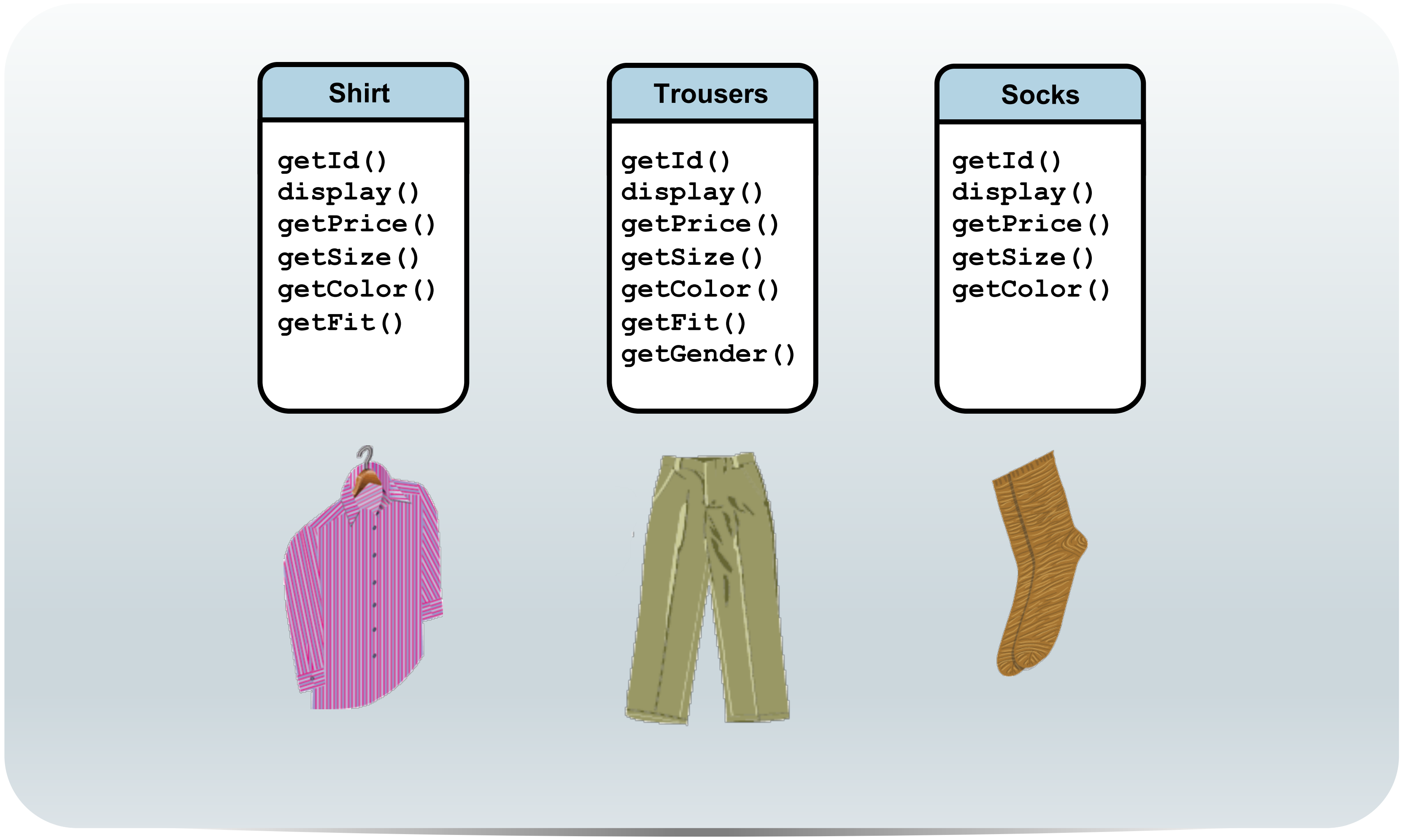
Shirt	**Trousers**

getId()
getPrice()
getSize()
getColor()
getFit()
	getId()
	getPrice()
	getSize()
	getColor()
	getFit()
	getGender()

setId()
setPrice()
setSize()
setColor()
setFit()
	setId()
	setPrice()
	setSize()
	setColor()
	setFit()
	setGender()

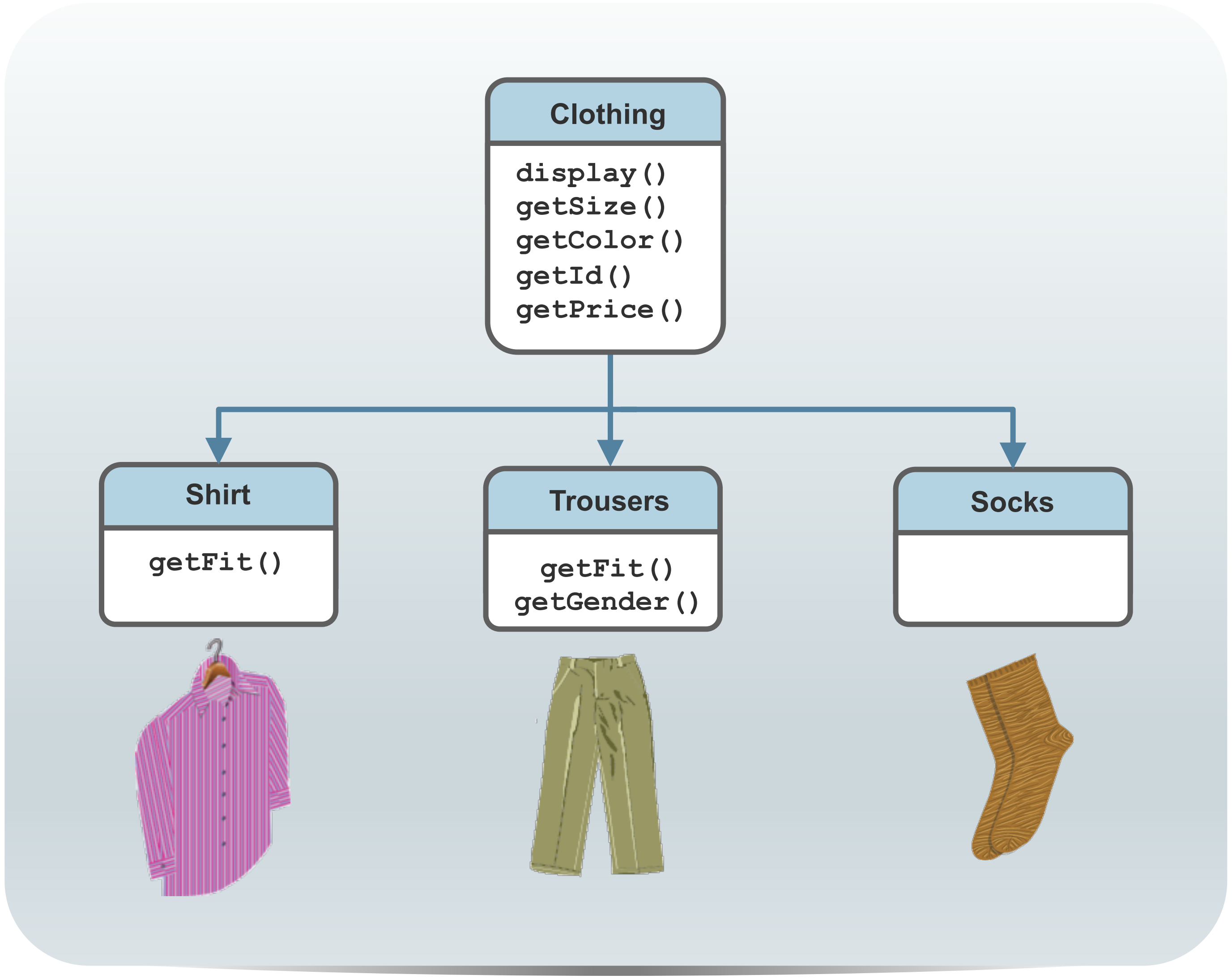
display()	display()

Code Duplication



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

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

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

Use the `extends` keyword.

```
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

```
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

```java
public class Clothing {
  // fields given default values
  private int itemID = 0;
  private String desc = "-description required-";
  private char colorCode = 'U';
  private double price = 0.0;

  // Constructor
  public Clothing(int itemID, String desc, char color,
     double price ) {
      this.itemID = itemID;
      this.desc = desc;
      this.colorCode = color;
      this.price = price;
  }
}
```

Shirt Class: Part 1

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

  public Shirt(int itemID, String description, char
          colorCode, double price, char fit) {
       super (itemID, description, colorCode, price);

       this.fit = fit;
  }
  public char getFit() {
      return fit;
  }
  public void setFit(char fit) {
      this.fit = fit;
  }
```

Reference to the superclass constructor

Reference to this object

Constructor Calls with Inheritance

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

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

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

```
public class Clothing{
  private double price;

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

Inheritance and Overloaded Constructors

```
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;          }}
```

```
public class Clothing{
  private double price;

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

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

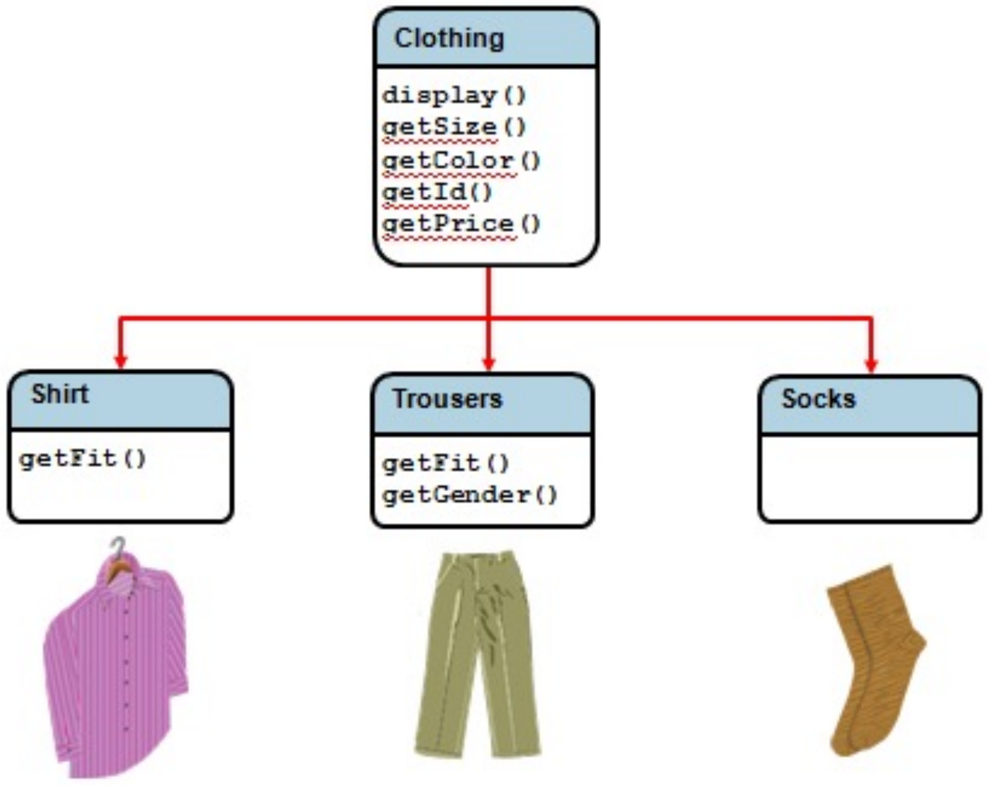
Overriding: A subclass implements a method that already has an implementation in the superclass.

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

```
public void display() {
  System.out.println("Item ID: " + getItemID());
  System.out.println("Item description: " + getDesc());
  System.out.println("Item price: " + getPrice());
  System.out.println("Color code: " + getColorCode());
 }
 public String getDesc (){
     return desc;
 }
 public double getPrice() {
     return price;
 }
 public int getItemID() {
     return itemID;
 }
 protected void setColorCode(char color){
     this.colorCode = color; }
```

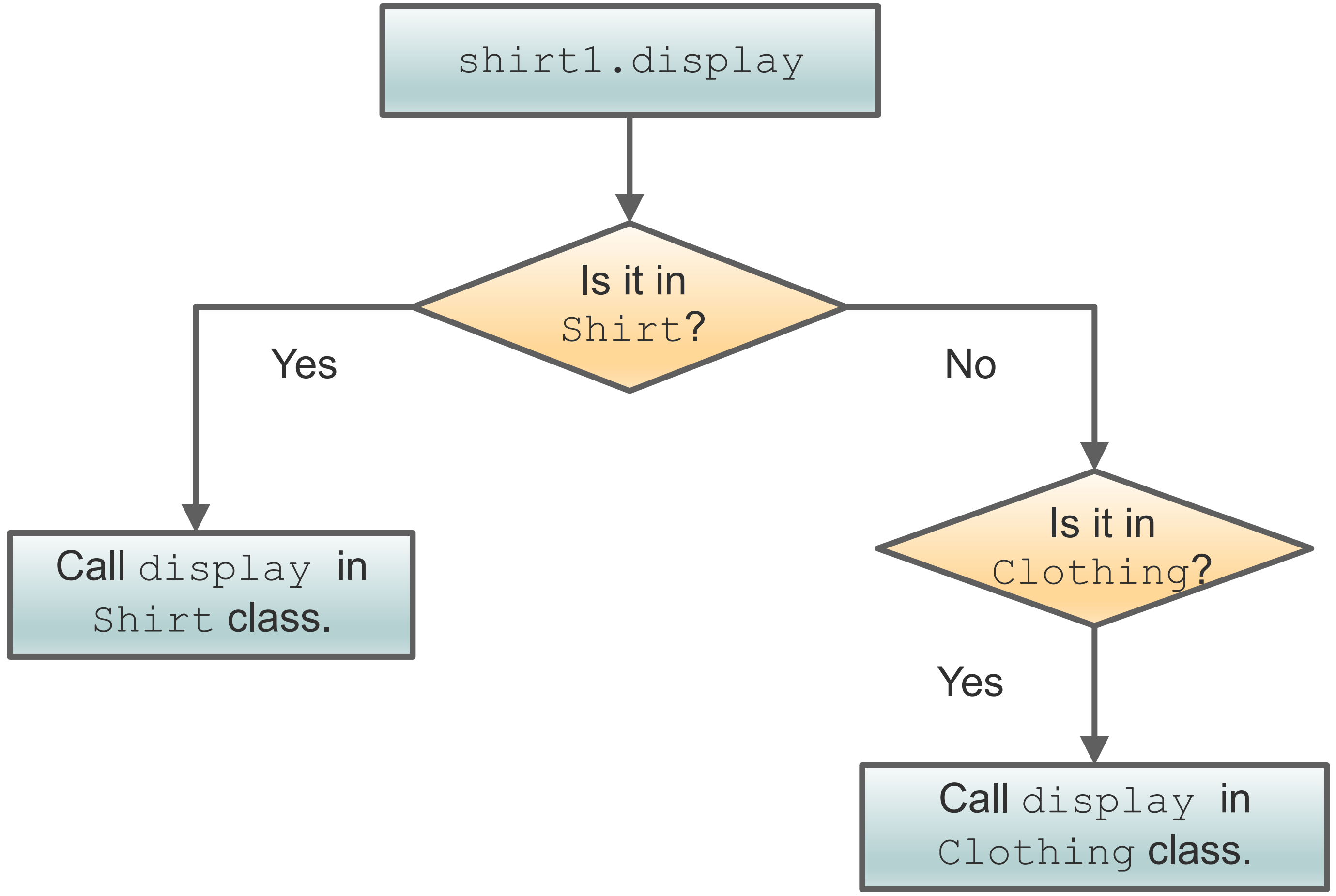
Assume that the remaining `get/set` methods are included in the class.

`Shirt` Class: Part 2

```java
 // These methods override the methods in Clothing
public void display() {
    System.out.println("Shirt ID: " + getItemID());
    System.out.println("Shirt description: " + getDesc());
    System.out.println("Shirt price: " + getPrice());
    System.out.println("Color code: " + getColorCode());
    System.out.println("Fit: " + getFit());
 }

 protected void setColorCode(char colorCode) {
    //Code here to check that correct codes used
    super.setColorCode(colorCode);
 }
}
```

Call the superclass's version of `setColorCode`.

Overriding a Method: What Happens at Run Time?



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

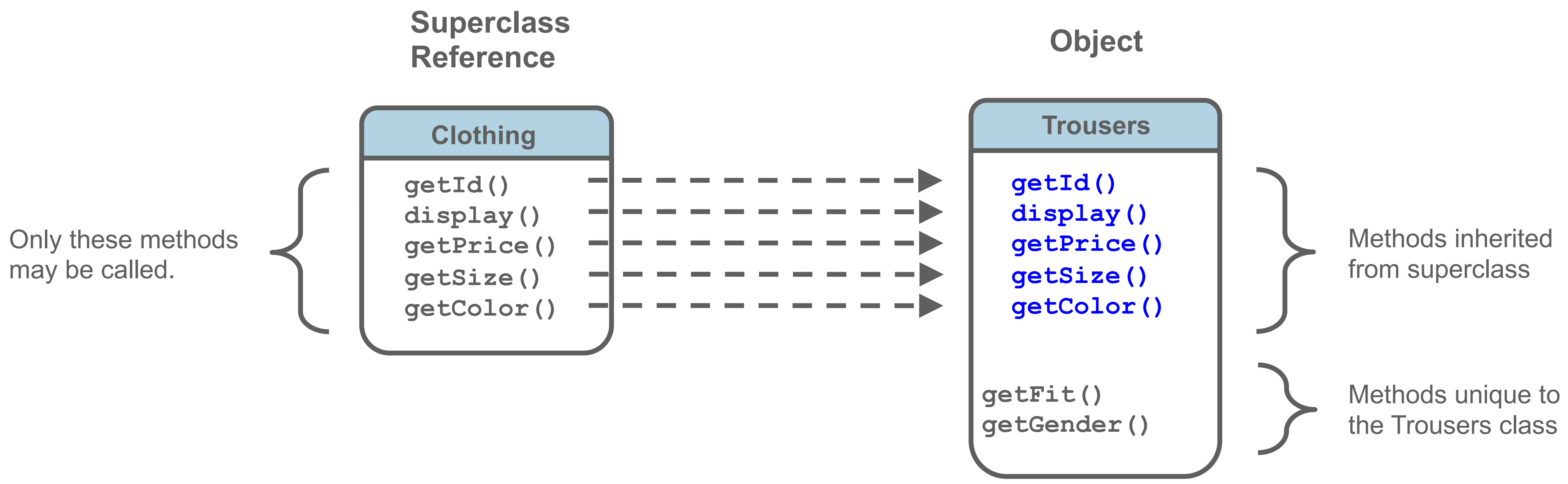
```
Clothing c1 = new ??();

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:

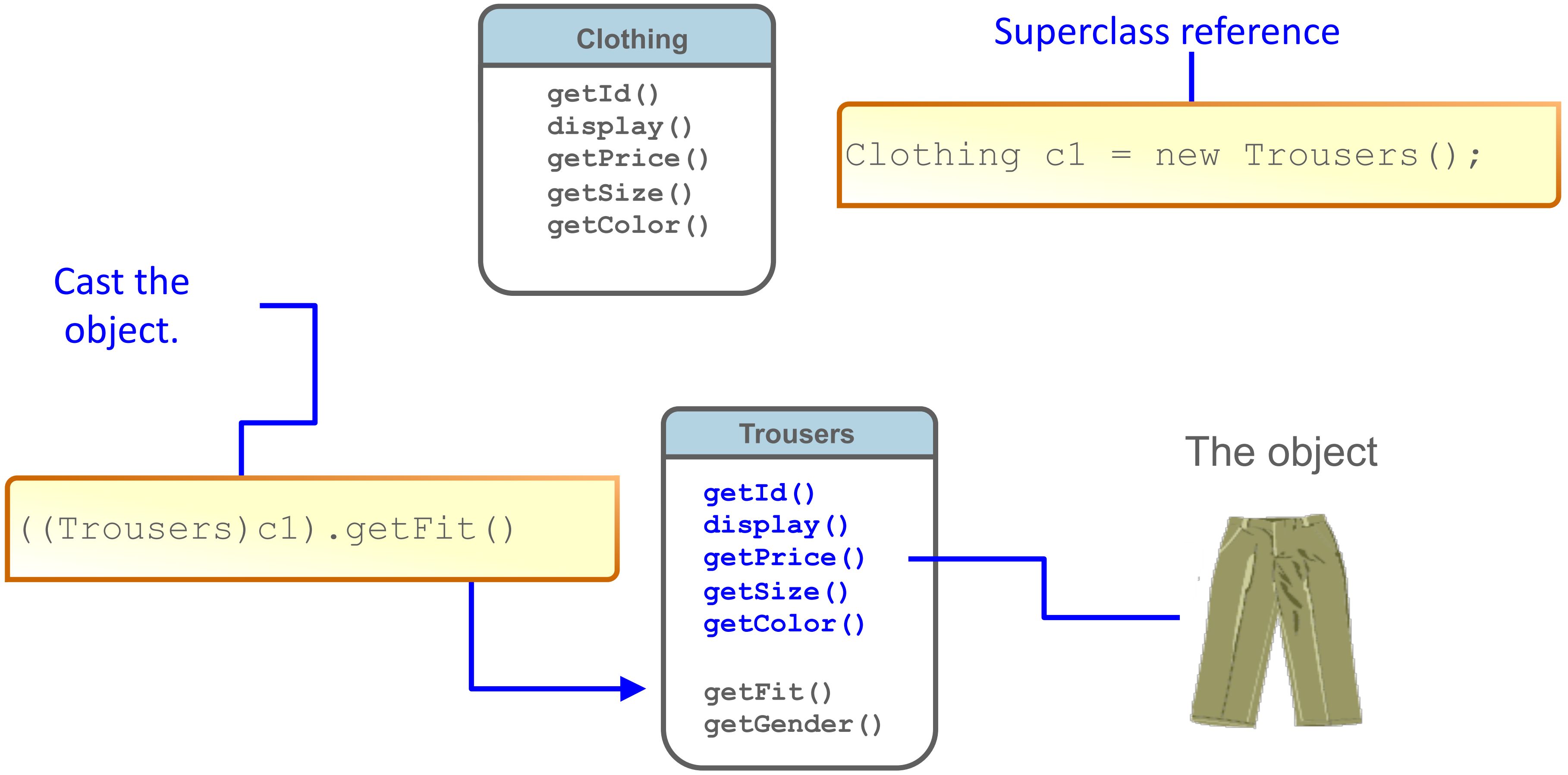
- Trousers objects show more fields in the display method.
- Different subclasses accept a different subset of valid color codes.

Accessing Methods Using a Superclass Reference



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

Casting the Reference Type



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

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

instanceof Operator

Possible casting error:

```
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:

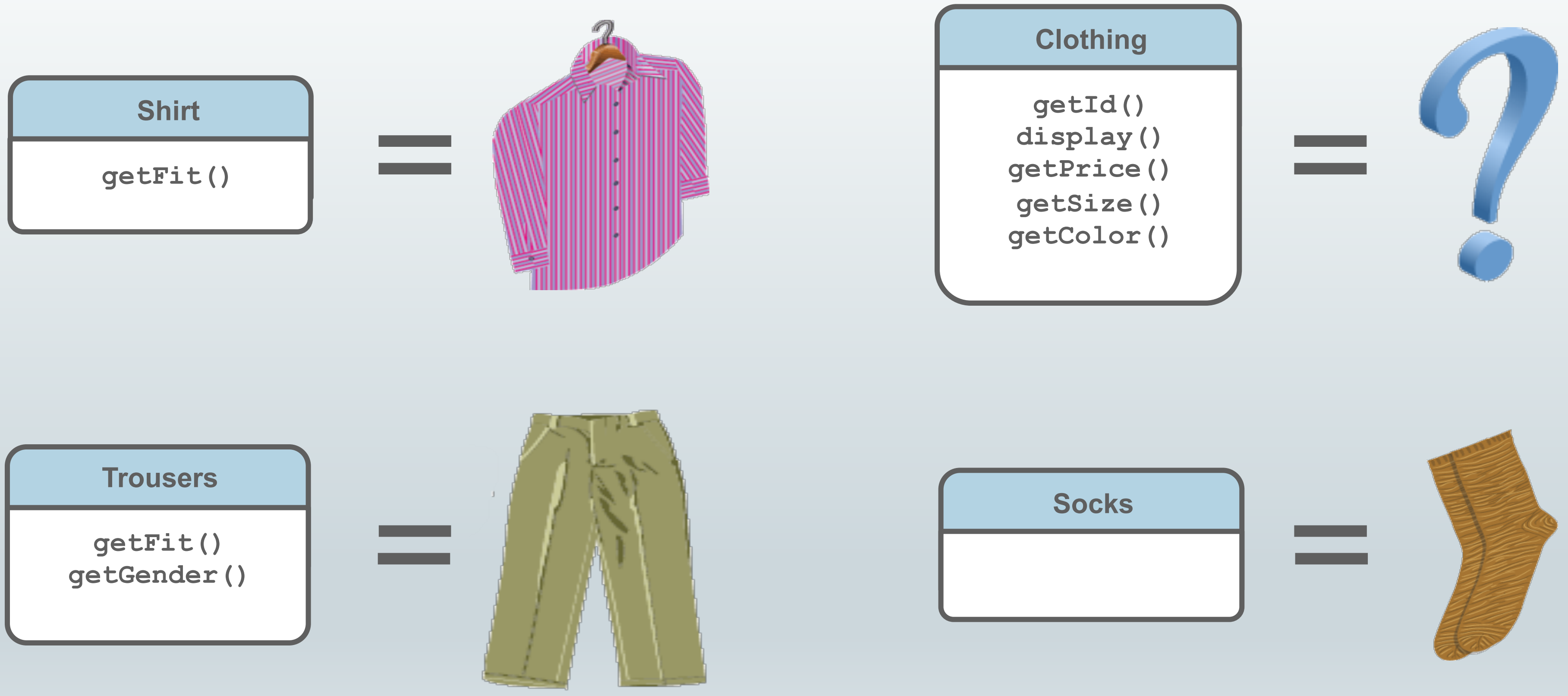
```
public static void displayDetails(Clothing cl) {
   cl.display();
   if (cl instanceof Trousers) {
       char fitCode = ((Trousers)cl).getFit();
       System.out.println("Fit: " + fitCode);
   }
   else { // Take some other action }
```

instanceOf returns true if c1 is a Trousers object.

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(){
        return id;
    }

    public abstract double getPrice();
    public abstract void display();
}
```

Concrete method

Abstract methods

Extending Abstract Classes

```
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
```

```
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