Inheritance

Josh Wilcox (jw14g24@soton.ac.uk)

February 11, 2025

Contents

1 Definition

- Allows classes to inherit some properties that it shares with other classess
  - Analogous to a common template of classes
  - Allows for shipping common functionality while maintaining encapsulation
- Any class that other classes are inherited from are known as a **superclass**
- Subclasses inherit all the properties and methods from the superclass
  - They can also add their own methods and properties

2 Inhertiance Hierarchies

- Subclasses can also be a superclass
- This means we can have a *tree of classes* called an **inheritance hierarchy**

3 Java Implementation

```java
public class Item
// No change to Item needs to be made to specify that it is a superclass
{
private String title;
private int playingTime;
private boolean gotIt;
private String comment;
// constructors and methods omitted.
}
```

```java
public class CD extends Item
// This class inherits from the superclass Item
{
private String artist;
private int numberOfTracks;
// constructors and methods omitted.
}
```

```java
public class DVD extends Item
// This class inherits from the superclass Item
{
private String director;
// constructors and methods omitted.
}
```

4 Encapsulation and Inheritance

4.1 Encapsulation

- Encapsulation is the principle that every class should be responsible for itself
  - This can be enforced using the `public` and `private` keywords

4.2 How does Encapsulation work in an Inheritance Heirarchy

- The question to ask is Should sub-classes be able to see all the properties and methods in their super classes

4.3 More Access Modifiers

4.3.1 Public

- Everyone Can see it

4.3.2 Private

- Only *This Class* can see it

4.3.3 Protected

- Only *this class* and its *subclasses* and the *package* can see it

4.3.4 Default

- Only *this class* and the *package* can see it

5 Encapsulation and Constructors

- Constructors are the special method that are called on creation using te `new` keyword

5.1 super-class constructor

- In Java, the `super()` keyword is used to call the constructor of the superclass.
- This is useful for initializing the state of the superclass when creating an instance of a subclass.
- The call to `super()` **must be the first statement in the subclass** constructor.
- If the superclass constructor requires arguments, they must be provided in the `super()` call.

```
public class Item {
    private String title;
    private int playingTime;
    private boolean gotIt;
    private String comment;

    public Item(String title, int playingTime) {
        this.title = title;
        this.playingTime = playingTime;
        this.gotIt = false;
        this.comment = "";
    }
}
```

```
public class CD extends Item {
    private String artist;
    private int numberOfTracks;

    public CD(String title, int playingTime, String artist, int numberOfTracks) {
        super(title, playingTime); // Call to superclass constructor
```

```
        this.artist = artist;
        this.numberOfTracks = numberOfTracks;
    }
}
```

- Encapsulation is maintained because the superclass's private fields are not directly accessible by the subclass.
- The subclass can only interact with the superclass's fields through the superclass's public or protected methods.
- This ensures that the internal state of the superclass is protected and can only be modified in controlled ways.

6 The Object Class

- All items inherit from `Object`
- This allows for the default constructor and the `.equals()` method
- It also allows to pass any object to `System.out.println()`