16.1

```
ngnlesn33 *
public static void main(String[] args) {
    List<Media> mediae = new ArrayList<Media>();
    DigitalVideoDisc dvd1 = new DigitalVideoDisc( id: 1, title: "The Lion King", category: "Animation", director: "Son", length: 5, cost: 19.95f);
    ArrayList<String> authors = new ArrayList<>(List.of( e1: "J.K. Rowling"));
    Book book1 = new Book( id: 1, title: "Harry Potter and the Sorcerer's Stone", category: "Fantasy", cost: 24.95f, authors);
    ArrayList<String> trackList = new ArrayList<>(Arrays.asList("Beat It", "Thriller"));
    CompactDisc cd1 = new CompactDisc( id: 2, title: "Thriller", category: "Pop", cost: 15.95f, length: 42, director: "Michael Jackson", artist: "Micha

    mediae.add(dvd1);
    mediae.add(book1);
    mediae.add(cd1);

    for (Media m : mediae) {
        System.out.println(m.toString());
    }
}
```

The output:

```
DVD - The Lion King - Animation - Son - 5: 19.95 $
AimsProject.src.hust.soict.hedspi.aims.media.Book@7f31245a
AimsProject.src.hust.soict.hedspi.aims.media.CompactDisc@6d6f6e28
```

Explain: The output you're seeing is the result of the toString() method being called on each object in the mediae list. The first line is the output from the DigitalVideoDisc object's toString() method, which has been overridden to display the DVD's details in a readable format: "DVD - The Lion King - Animation - Son - 5: 19.95 $".

The other two lines are the default toString() method outputs from the Object class for the Book and CompactDisc objects, respectively. Since these classes have not overridden the toString() method, the output is the class name followed by the object's hashcode in hexadecimal.

Reading Assignment:

Advantages of Polymorphism:

- Flexibility: Allows methods to be written that don't need to change if new subclasses are created.
- Simplifies Code: Reduces the complexity of the code by allowing one method to handle different data types and classes.
- Maintainability: Makes it easier to maintain and extend the code as new subclasses can use existing methods with polymorphic behavior.

Inheritance and Polymorphism:

- Foundation for Polymorphism: Inheritance allows a subclass to inherit methods from a parent class, which can be overridden to provide specific behaviors, enabling polymorphism.
- Code Reusability: Facilitates code reuse by inheriting methods and attributes, while polymorphism allows these methods to be used in a more general way.

Differences between Polymorphism and Inheritance:

- Conceptual Difference: Inheritance is a mechanism for creating a new class from an existing class, while polymorphism is the ability of different classes to respond to the same method call in different ways.
- Functionality: Inheritance creates a hierarchical relationship between classes, whereas polymorphism allows for the use of a single interface to represent different underlying forms (data types).

17.

Answer:

- Class Implementing Comparable: The class that should implement the `Comparable` interface is the `Media` class, as it is the base class for all media items in the cart, including DVDs, books, and CDs.

- Implementing compareTo(): In the `Media` class, you would implement the `compareTo()` method to reflect the desired ordering. For example, to sort by title then cost, you could implement it as follows:

```
@Override
public int compareTo(Media other) {
    int titleComparison = this.getTitle().compareTo(other.getTitle());
    if (titleComparison != 0) {
        return titleComparison;
    }
    // If titles are the same, compare by cost (lower cost first)
```

```
    return Float.compare(this.getCost(), other.getCost());
}
```

- Multiple Ordering Rules: With the `Comparable` interface, you can only define one natural ordering. Therefore, you cannot have two ordering rules (by title then cost and by cost then title) using this approach directly. You would need to use additional `Comparator` classes for alternative orderings.

- Different Ordering for DVDs: If DVDs have a different ordering rule, you would override the `compareTo()` method in the `DigitalVideoDisc` subclass to reflect this new rule. Here's an example:

```
@Override
public int compareTo(Media other) {
    if (other instanceof DigitalVideoDisc) {
        DigitalVideoDisc otherDVD = (DigitalVideoDisc) other;
        int titleComparison = this.getTitle().compareTo(otherDVD.getTitle());
        if (titleComparison != 0) {
            return titleComparison;
        }
        // For DVDs, compare by decreasing length
        int lengthComparison = -Integer.compare(this.getLength(),
otherDVD.getLength());
        if (lengthComparison != 0) {
            return lengthComparison;
        }
        // If titles and lengths are the same, compare by cost (lower cost first)
```

```
        return Float.compare(this.getCost(), otherDVD.getCost());
    } else {
        // If not comparing with another DVD, use the Media class's compareTo
        return super.compareTo(other);
    }
}
```

In this example, the `compareTo()` method in `DigitalVideoDisc` first checks if the other object is a DVD, then compares by title, then by decreasing length, and finally by cost. If the other object is not a DVD, it falls back to the `compareTo()` method defined in the `Media` class. This allows DVDs to have a custom sorting rule while still being able to be compared with other media types.