Composition & Interfaces

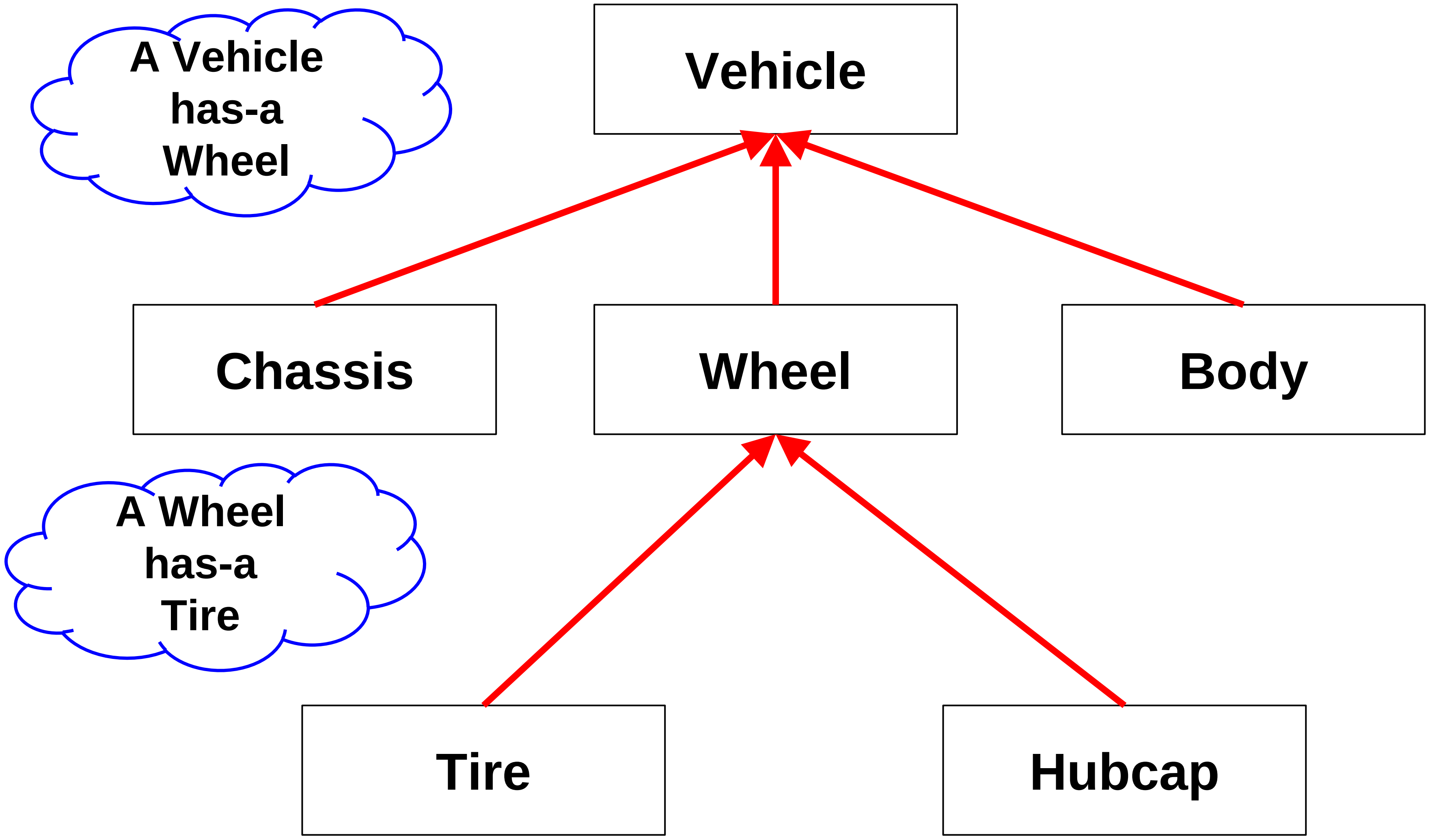
AP Computer Science

Composition

Composition

- Composition is when one object is composed of another object.
- This is defined as a **has-a relationship**.
  - A person **has-a** leg
  - A student **has-a** grade
  - A shoe **has-a** lace

Hierarchy Example



Composition Example

What is the Shape class composed of?

```
public class Shape {
   private int x, y;
   // constructors not shown
   // modifiers and accessors for y not shown
   public void setX(int xPos){
      x = xPos;
   }
   public int getX(){
      return x;
   }
}
```

Can you think of another way to organize this composition? What do x and y represent?

Composition with Objects

```
public class Point {
   private int x, y;

   public Point(int xPos, int yPos){
      x = xPos;
      y = yPos;
   }

   // modifiers and accessors for y not shown
   public void setX(int xPos){
      x = xPos;
   }

   public int getX(){
      return x;
   }
}
```

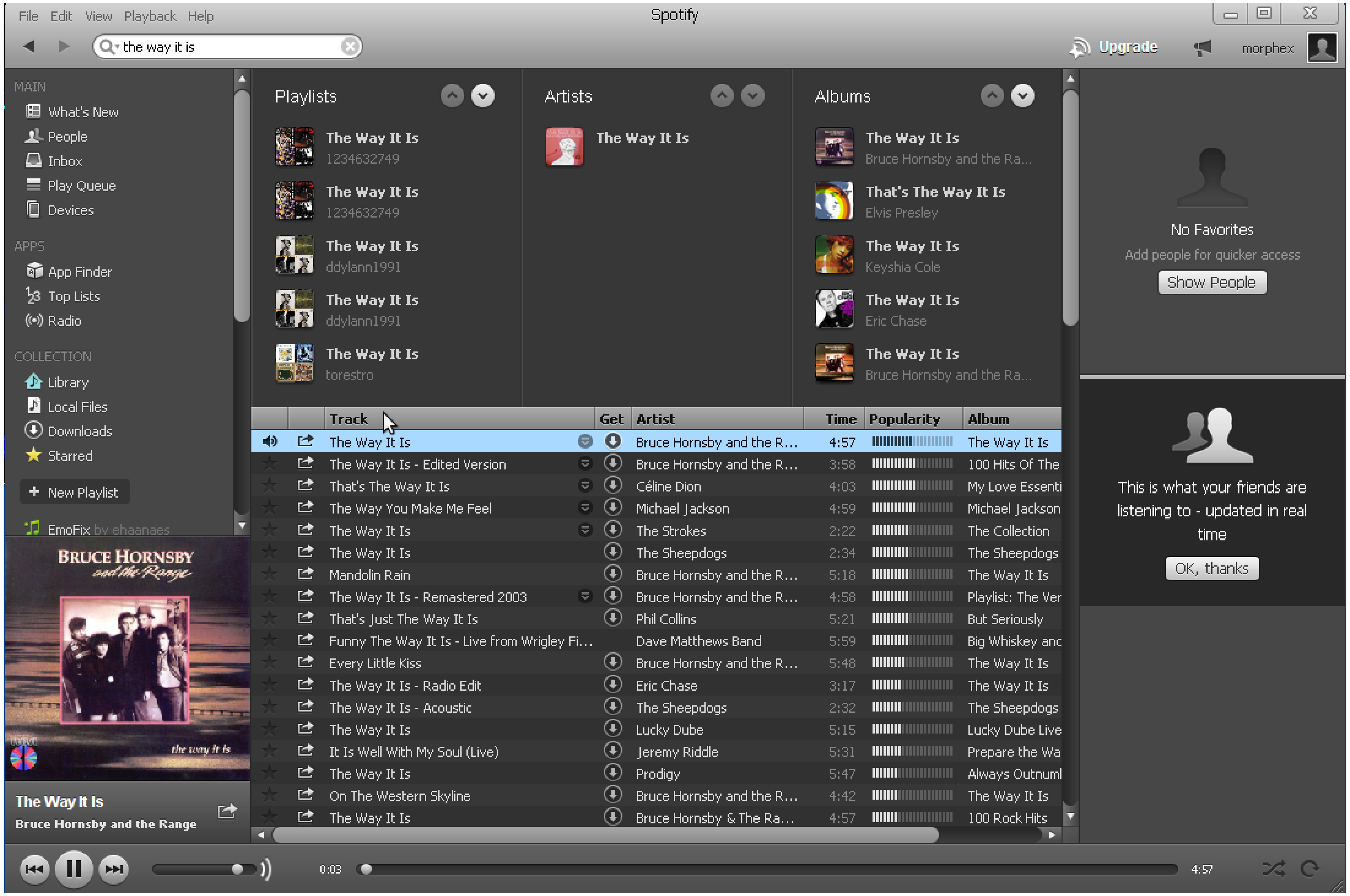
Composition with Objects

```
public class Shape {
   private Point position;  //a Shape has-a Point
   public Shape(){
      this(0, 0);
   }
   public Shape(int xPos, int yPos){
      position = new Point(xPos, yPos);
   }}
```

```
public class Point {
   private int x, y;
   public Point(int xPos, int yPos){
      x = xPos;
      y = yPos;
   }
   // modifiers and accessors not shown
}
```

Interfaces

Interfaces



This is what your friends are listening to - updated in real time

Playlists

- The Way It Is — 1234632749
- The Way It Is — 1234632749
- The Way It Is — ddylann1991
- The Way It Is — ddylann1991
- The Way It Is — torestro

Artists

- The Way It Is

Albums

- The Way It Is — Bruce Hornsby and the Ra...
- That's The Way It Is — Elvis Presley
- The Way It Is — Keyshia Cole
- The Way It Is — Eric Chase
- The Way It Is — Bruce Hornsby and the Ra...

Track	Get	Artist	Time	Popularity	Album
The Way It Is		Bruce Hornsby and the R...	4:57		The Way It Is
The Way It Is - Edited Version		Bruce Hornsby and the R...	3:58		100 Hits Of The
That's The Way It Is		Céline Dion	4:03		My Love Essenti
The Way You Make Me Feel		Michael Jackson	4:59		Michael Jackson
The Way It Is		The Strokes	2:22		The Collection
The Way It Is		The Sheepdogs	2:34		The Sheepdogs
Mandolin Rain		Bruce Hornsby and the R...	5:18		The Way It Is
The Way It Is - Remastered 2003		Bruce Hornsby and the R...	4:58		Playlist: The Ver
That's Just The Way It Is		Phil Collins	5:21		But Seriously
Funny The Way It Is - Live from Wrigley Fi...		Dave Matthews Band	5:59		Big Whiskey anc
Every Little Kiss		Bruce Hornsby and the R...	5:48		The Way It Is
The Way It Is - Radio Edit		Eric Chase	3:17		The Way It Is
The Way It Is - Acoustic		The Sheepdogs	2:32		The Sheepdogs
The Way It Is		Lucky Dube	5:15		Lucky Dube Live
It Is Well With My Soul (Live)		Jeremy Riddle	5:31		Prepare the Wa
The Way It Is		Prodigy	5:47		Always Outnuml
On The Western Skyline		Bruce Hornsby and the R...	4:42		The Way It Is
The Way It Is		Bruce Hornsby & The Ra...	4:57		100 Rock Hits

The Way It Is
Bruce Hornsby and the Range

Interfaces

```
public interface Spotify {
   public void pickSong(Song song);
   public void play();
   public void next();
   public void previous();
}
```

- Interfaces provide a very simple view of how a program should behave.
- Actual implementations may be very complex.
- This is an example of **abstraction**: a concept or idea not associated with any specific implementation.

Interfaces

- If your class uses an interface, it **must** implement all methods defined in that interface.
- Each **method signature** must match the interface exactly.
  - Return type
  - Method name
  - Parameters
- Interfaces have no constructors and cannot be instantiated

Shapes Using an Interface

```
public interface Area {
   public double getArea();
}
```

```
public class Circle implements Area{
   private double radius;
   public Circle(double r) { radius = r; }
   public double getArea(){
   return Math.PI * radius * radius;
   }
}
```

Shapes Using an Interface

```
public interface Area {
   public double getArea();
}
```

```
public class Shape {
   private int x, y;
   public Shape() { x = y = 0; }
}
```

```
public class Circle extends Shape implements Area {
   private double radius;
   public Circle(double r) { radius = r; }
   public double getArea(){
      return Math.PI * radius * radius;
   }
}
```

Interfaces with Variables

```
public interface Area {
   public double getArea();
   public static final double PI = Math.PI;
}
```

```
}
```

- Interfaces may also contain variables, which are always `public static final`
  - `static` - initialized only **once** and **shared** by all objects instantiated of that class
  - `final` - cannot change, i.e. constant
    - Therefore, they must be defined and assigned a value in the interface
- Java automatically appends `public`, `static`, and `final`, so you do not need to write them

Interfaces with Variables

```
public interface Area {
   public double getArea();
   public static final double PI = Math.PI;
}
```

```
public class Shape {
   private int x, y;
   public Shape() { x = y = 0; }
}
```

```
public class Circle extends Shape implements Area {
   private double radius;
   public Circle(double r) { radius = r; }
   public double getArea(){
      return PI * radius * radius;
   }}
```

The Comparable Interface

```
public interface Comparable {
   int compareTo(Object obj);
}
```

- If you implement this interface, your **`compareTo`** method must follow these rules:
  - Compares this object with **`obj`**
  - Returns a negative integer, zero, or a positive integer, when this object is less than, equal, or greater than **`obj`**, respectively

The CompareTo() Method

```
public class Circle extends Shape implements Comparable {
   private double radius;
   public Circle(int x, int y, double r) {/*not shown */}
   public double getRadius() { return radius; }
   public int compareTo(Object obj) {
      Circle temp = (Circle)obj;
      if(getRadius() == temp.getRadius())
         return super.compareTo(temp);
      if(getRadius() < temp.getRadius())
         return -1;
      if(getRadius() > temp.getRadius())
         return 1;
}}
```

Note the differences between this slide and the next slide.

The CompareTo() Method

```
public class Circle extends Shape implements
   Comparable<Circle> {
   private double radius;
   public Circle(int x, int y, double r) {/*not shown */}
   public double getRadius() { return radius; }
   public int compareTo(Circle temp) {
      if(getRadius() == temp.getRadius())
         return super.compareTo(temp);
      if(getRadius() < temp.getRadius())
         return -1;
      if(getRadius() > temp.getRadius())
         return 1;
}}
```

Note the differences between this slide and the previous slide.

The CompareTo() Method

```java
// main method
Circle one = new Circle(50, 50, 5);
Circle two = new Circle(25, 25, 8);
Circle three = new Circle(25, 25, 10);
Circle four = new Circle(50, 50, 5);
System.out.println(one.compareTo(two));
System.out.println(two.compareTo(three));
System.out.println(three.compareTo(four));
System.out.println(one.compareTo(four));
```

Output

```
-1
-1
1
0
```

- The first two parameters are the x and y coordinates and the last parameter is the radius.
- The radius is compared first and then the x and y coordinates.

Interfaces

- A subclass **`extends`** only one superclass
  - Java uses single inheritance
- A subclass **`implements`** one or more interfaces
  - This provides the benefits of multiple inheritance
- An subinterface **`extends`** one or more super interfaces

```
public class Circle implements Area, Comparable {
  // implementation not shown.
}
```

}

Polymorphism

- If a class implements an interface the reference can be of the type of the Interface
  - `Comparable cir = new Circle();`
- You can call the compareTo method defined in **`Comparable`**, but no other methods without a cast
- This is an example of **polymorphism**, i.e. the ability of the **`Comparable`** object to take on multiple forms

Polymorphism

What is the output?

```
ArrayList<Comparable> list;
list = new ArrayList<Comparable>();
list.add("zebra");
list.add("monkey");
list.add("lion");
Collections.sort(list);
System.out.println(list.toString());
```

Output

```
[lion, monkey, zebra]
```

- This would sort the list based on the compareTo method from String
- All items in this case need to be the same type