Improving Structure with inheritance

- Define one superclass
- Define subclasses
- The superclass defines common attributes
- The subclasses inherit the superclass attributes
- The subclasses add their own attributes

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

    // constructors and methods omitted.
}
// Subclass
public class CD extends Item
{
    private String artist;
    private int numberOfTracks;

    // constructors and methods omitted.
}
```

1 Superclass constructor call

- Subclass constructors must always contain a ”super” call
- If none is written, the compiler inserts one (without parameters)
- Must be the first statement in the subclass constructor

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

    /**
    * Constructor for objects of class CD
    */
    public CD(String theTitle, String theArtist, int tracks, int time)
    {
        super(theTitle, time);
        artist = theArtist;
        numberOfTracks = tracks;
    }

    // methods omitted
}
```

2 Subclasses and subtyping

- Classes define types
- Subclasses define subtypes
- Objects of subclasses can be used where objects of supertypes are used (this is called substitution)

3 Polymorphic variables

- Object variables in Java are polymorphic (they can hold objects of more than one type)
- They can hold objects of the declared type, or of subtypes of the declared type

4 Casting

- Can assign subtype to supertype
- Can't assign supertype to subtype

For example, the last line of this causes a compile time error

```
Vehicle v;
Car c = new Car();
v=c; // this is fine
c=v; // this causes a compile time error
```

This can be fixed with casting like so

```
c= (Car) v;
```

- An object type in parentheses
- Used to overcome "type loss"
- The object is not changed in any way
- A runtime check is made to ensure the object really is of that type

5 Polymorphic collections

- All collections are polymorphic
- The elements are of type object

6 Collections and primitive types

All objects can be entered into collections because collections accept elements of type Object and all classes are subtypes of Object

7 Wrapper classes

- Primitive types (int, char, etc) are not objects. They must be wrapped into an object

Simple Type	Wrapper Class
int	Integer
float	Float
char	Character

```
int i=18;
Integer iwrap = new Integer(i); // wrapping the value
itn value = iwrap.intValue(); //unwrap it
```

8 Static type and dynamic type

- The declared type of a variable is its static type
- The type of the object a variable refers to is its dynamic type
- The compiler's job is to check for static-type violations

9 Overriding

- Superclass and subclass define methods with the same signature
- Each has access to the fields of its class
- Superclass satisfies static type check
- Subclass method is called at runtime - it overrides the superclass version

10 Method lookup

- The variable is accessed
- The object stored in the variable is found
- The class of the object is found
- The class is searched for a method match
- If no match is found, the superclass is searched
- This is repeated until a match is found, or the class hierarchy is exhausted
- Overriding methods take precedence

11 Super call in methods

- Overriden methods are hidden, but we often still want to be able to call them
- An overriden method can be called from the method that overrides it
  - `super.method(...)`

12 Method polymorphism

- Method calls are polymorphic - the actual method called depends on the dynamic object type

13 The Object class' methods

- Methods in the objects are inherited by all classes
- Any of those may be overriden
- The `toString` method is commonly overriden:
  - `public String toString()`
  - Returns a string representation of the object
  - The programmer might want a custom representation of the data as a string
  - Note that calls to println with just an object automatically result in toString being called
- Equals can also be overridden if the manner in which two things equal each other is not the traditional definition

14 Protected access

- Private access in the superclass may be too restrictive for a subclass
- The closer inheritance relationship is supported by protected access
- Protected access is more restricted than public access
- It is still recommended to keep fields provided and to define protected accessors and mutators