



Object Oriented Programming

Problem Set: Assignment #1 due Thursday, February 9

D - 1



Object-Oriented Programming (OOP)

- In OOP, computations are described in terms of collections of stateful **objects** that communicate by passing **messages** to one other.
- Objects are like actors in a play; the programmer is the playwright & director.
- A **class** specifies the behavior (variables and methods) of a collection of objects = the **instances** of the class.
- Related classes can be organized into **inheritance hierarchies**, which allow one class to extend and/or override the variables and methods of other classes.
- OOP is one of several programming paradigms. Others include imperative programming, function-oriented programming, logic programming. CS251 Programming Languages covers these.

D - 2



Today's Play: Songs in a Music Library

You need to keep track of a (initially unknown) group of songs in a music library.

Each song has
a title, an artist, an album, a duration

You need to gather information about a new song,
give it a title, update its information

You also need to keep track of some
statistics and comparisons

D - 3



The Anatomy of a Java Class

A Java class can contain **5** kinds of declarations:

- **2** kinds of variable declarations:
 - an **instance variable** for every instance of the class
 - a **class (static) variable** in exactly one place (the class itself)
- **3** kinds of method declarations.
 - A **constructor method** specifies how to create a new instance. Via **this**, it can refer to all (including private) instance variables of the new instance that are declared in the method's class.
 - An **instance method** specifies how an instance (the **receiver = this**) responds to a message. It can also refer to all of the receiver's instance variables that are declared in the method's class. Note that the receiver is really just an argument with a special name (**this**).
 - A **class method** is a receiverless message that corresponds to functions/procedures in other languages. It cannot refer to **this**, since there is no receiver.

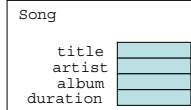
D - 4



Instance and Class Variables

```
public class Song {
  // instance variables: things about a specific song
  private String title;
  private String artist;
  private String album;
  private double duration; // in seconds

  // class variables: things about all songs
  private static int numSongs = 0;
  private static double avgDuration = 0.0;
  ...
}
```



D-5



Information Flow: Public/Protected/Private

Within a class *C*, any phrase (statement/expression) *P* can access:

- public class variables of any class;
- protected class variables of classes in the same package as *C*;
- private class variables of *C*;
- public instance variables of any object mentioned in *P*;
- protected instance variables of any object *O* mentioned in *P*, where *O*'s class is in the same package as *C*;
- private instance variables of any instance of *C*.

Additionally:

- Any method body can access all parameter variables.
- A constructor method body can access the newly constructed instance via **this**.
- An instance method body can access the receiver object via **this**.

D-6



When to use private and protected?

Private should be the default:

- This helps to preserve invariants
- Gives flexibility to implementer to change representations under the hood.

Protected is used for instance/class variables that need to be accessed by other classes in the same package, particularly subclasses of the given class.

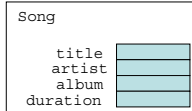
Public should only be used for instance/class variables as public when (1) their implementation will never change and (2) it's OK for the whole world to see/change their value. (Can prevent changing public variables like `Color.red` by declaring them `final`.)

D-7



Constructors

```
public class Song {
  ...
  /** 2-arg constructor: how to create a song titled title with a
  given duration*/
  public Song (String title, int duration) {
    this.title = title;
    this.artist = "unknown";
    this.album = "unknown";
    this.duration = duration;
    double totalDuration = avgDuration * numSongs;
    numSongs++;
    avgDuration = (totalDuration + duration)/numSongs;
  }
}
```



- If there is no confusion about instance variables and input parameters, the use of "this" is optional.

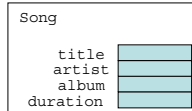
- Use it to initialize a new `Song` instance
`Song oldie = new Song("Respect", 145.0);`

D-8



More Constructors

```
public class Song {
    ...
    /** 4-args constructor: for a song you know a lot about */
    public Song (String title, String artist, String album,
                double duration) {
    }
}
```



Notice **OVERLOADING** of method name. To use it:

```
Song oldie2 = new Song("Respect", "Aretha Franklin", "Aretha's Gold", 145.0);
```

The previous 2-argument constructor could be written as:

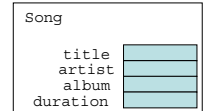
```
public Song (String title, int duration) {
    this(title, "unknown", "unknown", duration);
}
```

D-9



Instance get-Methods

```
public class Song {
    ...
    /** instance methods that return the values of the four instance
    * variables.
    */
    public String getTitle () {
    }
    public String getArtist () {
    }
    public String getAlbum () {
    }
    public double getDuration () {
    }
}
```

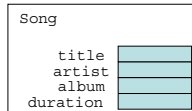


D-10



Instance set-Methods

```
public class Song {
    ...
    /** instance methods that change the values of the four instance
    * variables.
    */
    public void setTitle (String newTitle) {
    }
    public void setArtist (String newArtist) {
    }
    public void setAlbum (String newAlbum) {
    }
    public void setDuration (double newDuration) {
    }
}
```



To use it:

```
oldie.setArtist("Aretha Franklin");
```

But :

```
oldie.artist = "Aretha Franklin"; // ERROR: why?
```

D-11



Class Methods

```
public class Song {
    ...
    /** class method that returns the number of Song
    * objects created so far */
    public static int getNumSongs () {
    }
    /** class method that returns the average duration of
    * all the Songs objects created so far */
    public static double getAvgDuration () {
    }
}
```

Static means the variable was created once.

There may be many Song instances,
but only one numSongs variable and one avgDuration variable

D-12



toString() - a good method to have

```
public class Song {
    ...
    /** instance method that returns a string containing
     * all of the info stored in a single Song object */
    public String toString () {
        String S = "\nTitle:\t" + title + "\nArtist:\t" + artist;
        S = S + "\nAlbum:\t" + album + "\n Duration: " + duration;
        return S;
    }
    ...
}
```

\n is newline \t is tab

It is invoked by default:

```
System.out.println(oldie2);
```

Produces:

```
Title: Respect
Artist:Aretha Franklin
Album: Aretha's Gold
Duration: 145.0
```

D-13



One Way to Compare Instance Variables

```
public class Song {
    ...
    /** instance method that returns true if the Song object used to
     * invoke the method has shorter duration than the input Song object
     */
    public boolean hasShorterDurationThan (Song song2) {
        return this.duration < song2.duration;
    }
    ...
}
```

To use it:

```
Song prelude = new Song("Mass in B Minor", "Johann Sebastian Bach",
    "Bach to Bach", 241.0);
Song birthday = new Song("Happy Birthday", 48.0);
prelude.hasShorterDurationThan(birthday) => false
```

D-14



Another Way to Compare Instance Variables

```
public class Song {
    ...
    /** class method that returns true if song1 has shorter duration
     * than song2 */
    public static boolean shorterDuration (Song song1, Song song2) {
        ...
    }
    ...
}
```

To use it:

```
Song.shorterDuration(prelude, birthday) => false
```

D-15



Equality of Instances

```
public boolean equals (Song song2) {
    return ( (title.equals(song2.title))&&
        (artist.equals(song2.artist)) &&
        (album.equals(song2.album)) &&
        (duration == song2.duration));
}
```

It does not just compare instance references,
but examines all instance variables:

```
Song newBirthday = new Song("Happy Birthday", "Unknown", "Unknown", 48);
newBirthday.equals(birthday) => true
```

You decide what constitutes equality between instances.
e.g., it could mean equality between just a few instance variables

D-16



Array of Objects

```
/** creates an array of 5 Song objects, fills the array with
 * 5 new instances of the Song class, and then returns the array */
public static Song[] makeSongArray () {
    Song[] songArray = new Song[5];
    songArray[0] =
        new Song("Respect", "Aretha Franklin", "Aretha's Gold", 145);
    songArray[1] =
        new Song("Happy Birthday", "Unknown", "Unknown", 48);
    songArray[2] =
        new Song("Mass in B Minor", "Bach", "Bach to Bach", 241);
    songArray[3] =
        new Song("Jana Gana Mana", "Tagore", "Unknown", 105);
    songArray[4] =
        new Song("The Way I Are", "Timbaland", "Shock Value", 279);
    return songArray;
}
```

To use it:

```
Song[] songsToListenTo = makeSongArray();
```

D - 17



Operations on Array of Objects

```
/** searches the array of Song objects for a particular artist,
 * and if found, prints the titles of all songs performed by the artist.
 * If the artist is not found, a message is printed indicating this.
 */
public static void searchArtist (String artist, Song[] songs) {
    boolean found = false;
    for (int i = 0; i < songs.length; i++) {
        if (songs[i].getArtist().equals(artist)) {
            System.out.println(songs[i].getTitle() + " ");
            found = true;
        }
    }
    // Note that the loop is not terminated when the specified artist
    // is found, because an artist may perform multiple songs

    if (!found)
        System.out.println("No song found performed by that artist.");
}
```

D - 18



More Operations

```
/** returns the name of the Song with the longest duration */
public static String getLongestSong (Song[] songs) {
    int indexOfLongestSong = 0;
    for (int i = 1; i < songs.length; i++) {
        if (songs[indexOfLongestSong].hasShorterDurationThan(songs[i])) {
            indexOfLongestSong = i;
        }
    }
    return songs[indexOfLongestSong].getTitle();
}
```

To use it:

```
System.out.println(getLongestSong(songsToListenTo));
```

Change it to return the song object, not just its title...

What if the songs array is empty?

D - 19