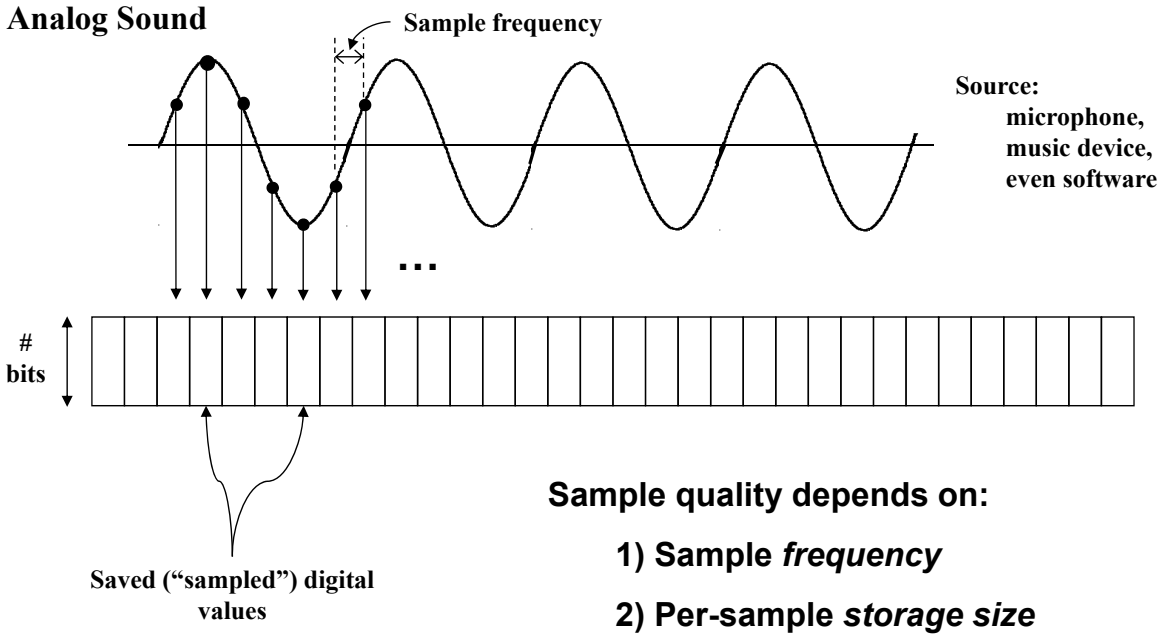
12 - Introduction to Sound

Computer Science Department
California State University, Sacramento

Overview

- **Sampled Audio**
- **Sound File Formats**
- **Popular Sound APIs**
- **Playing Sounds in CN1**
  - **Creating background sound that loops**

Sampled Audio



Sample quality depends on:

1) Sample *frequency*

2) Per-sample *storage size*

(bits per sample)

Sound File Formats

`.au`	Sun Audio File (Unix/Linux)
`.aiff`	Audio Interchange File Format (Mac)
`.cda`	CD Digital Audio (track information)
`.mpx`	MPEG Audio (mp, mp2, mp3, mp4)
`.mid`	MIDI file (sequenced, not sampled)
`.ogg`	Ogg-Vorbis file (open source)
`.ra`	Real Audio (designed for streaming)
`.wav`	Windows "wave file"

Finding sound files: `www.findsounds.com`

Example: WAVE Format

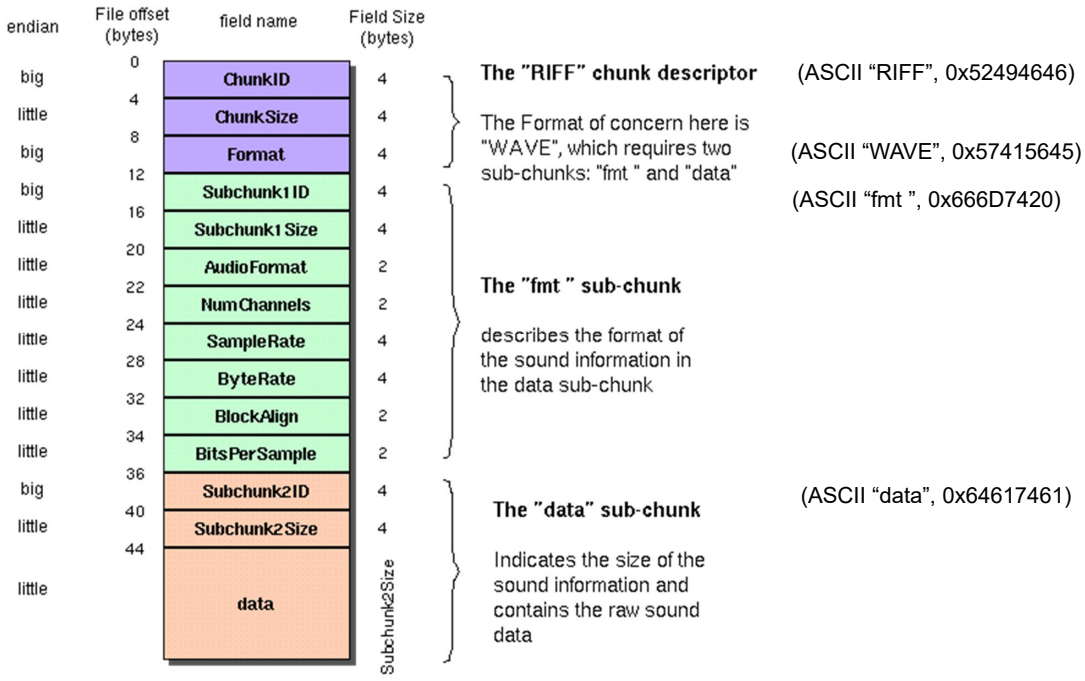


Image credit: http://ccrma.stanford.edu/courses/422/projects/WaveFormat/

Popular Sound API's

- Java **`AudioClip`** Interface
- JavaSound
- DirectSound / DirectSound3D
- Linux Open Sound System (OSS)
- Advanced Linux Sound Architecture (ALSA)
- OpenAL / JOAL

Java `AudioClip` Interface

- Originally part of web-centric **`Applets`**
- Supports
  - Automatic loading
  - **`play(), loop(), stop()`**
    - No way to determine progress or completion
- Supported sound file types depend on JVM
  - Sun default JVM: *.wav*, *.aiff*, *.au* , *.mid,* others…

Java Sound API

- A *package* of expanded sound support

```
import javax.sound.sampled;
import javax.sound.midi;
```

- New capabilities:
  - Skip to a specified file location
  - Control volume, balance, tempo, track selection, etc.
  - Create and manipulate sound files
  - Support for streaming
- Some shortcomings
  - Doesn't recognize some common file characteristics
  - Doesn't support spatial ("3D") sound

- "Open Audio Library"
  - 3D Audio API (**`www.openal.org`**)
- Open-source
- Cross-platform
- Modeled after OpenGL
- Java binding ("JOAL"):

  `www.jogamp.org`

Playing Sounds in CN1

- **`Media`** object should be created to play sounds.
- **`Media`** objects is created by the overloaded **`creatMedia()`** static method of the **`MediaManager`** class.
- **`createMedia()`** takes in an **`InputStream`** object which is associated to the audio file.
- **`Media`**, **`MediaManager`**, and **`InputStream`** are all built-in classes.

Important tips

- You must copy your sound files directly under the **`src`** directory of your project.
- You may need to refresh your project in your IDE (e.g., in Eclipse select the project and hit F5 OR right click on the project and select “Refresh”) for CN1 to properly locate the sound files newly copied to the **`src`** directory.
- **You must create sound (`Media`) objects after calling `show()`!**

Creating and playing a sound

```
import java.io.InputStream;
import com.codename1.media.Media;
import com.codename1.media.MediaManager;
...
/** This method constructs a Media object from the specified file, then plays the Media.*/
public void playSound (String fileName) {
    if (Display.getInstance().getCurrent() == null){
        System.out.println("Error: Create sound objects after calling show()!");
        System.exit(0);}
 try {
    InputStream is = Display.getInstance().getResourceAsStream(getClass(),
                                                        "/"+fileName);
    Media m = MediaManager.createMedia(is, "audio/wav");
    m.play();}
 catch (IOException e) {
    e.printStackTrace();}
}
//this method calls playSound() to play alarm.wav copied directly under the src directory
public void someOtherMethod(){
    playSound("alarm.wav")}
```

Encapsulating the sound

```
/** This class encapsulates a sound file as an Media inside a
 *  "Sound" object, and provides a method for playing the Sound.
 */`
public class Sound {
    private Media m;
    public Sound(String fileName) {
        if (Display.getInstance().getCurrent() == null){
           System.out.println("Error: Create sound objects after calling show()!");
           System.exit(0);}
        try{
        InputStream is = Display.getInstance().getResourceAsStream(getClass(),
                                                                   "/"+fileName);
        m = MediaManager.createMedia(is, "audio/wav");}
        catch(Exception e)
        {
           e.printStackTrace();}
    }
    public void play() {
        //start playing the sound from time zero (beginning of the sound file)
        m.setTime(0);
        m.play();}
}
```


Encapsulating the sound (cont)

- In the assignments, you should use encapsulated sounds.
- Create a single sound object for each audio file:

```
private Sound catCollisionSound, scoopSound;
public void createSounds() {
        catCollisionSound = new Sound("meow.wav");
        scoopSound = new Sound("scoop.wav");}
```

- Call `createSounds()` after `show()`!
- Operations that belong to the same type should play this single instance (e.g., make all cat-cat collisions call `catCollisionSound.play()`), instead of creating new instances.

Looping the Sound

- To create a sound which is played in a loop (e.g., the background sound), **`Media`** object **`m`** indicated above should be created differently.
- We must attach a **`Runnable`** object to it which is invoked when the media has finished playing.
- The **`run()`** method of the **`Runnable`** object must play the sound starting from its beginning.

Encapsulating Looping Sound

```
/**This class creates a Media object which loops while playing the sound */
public class BGSound implements Runnable{
  private Media m;
  public BGSound(String fileName){
   if (Display.getInstance().getCurrent() == null){
      System.out.println("Error: Create sound objects after calling show()!");
      System.exit(0);}
    try{
      InputStream is = Display.getInstance().getResourceAsStream(getClass(),
                                                          "/"+fileName);
      //attach a runnable to run when media has finished playing as the last parameter
      m = MediaManager.createMedia(is, "audio/wav", this);}
    catch(Exception e){
      e.printStackTrace();}
  }

  public void pause(){ m.pause();} //pause playing the sound

  public void play(){ m.play();} //continue playing from where we have left off

  //entered when media has finished playing
  public void run() {
    //start playing from time zero (beginning of the sound file)
    m.setTime(0);
    m.play();}
}
```

Use of Encapsulated Looping Sound

```
/**This form creates a looping sound and a button which pauses/plays the looping sound
 */

public class BGSoundForm extends Form implements ActionListener{

  private BGSound bgSound;
  private boolean bPause = false;
  public BGSoundForm() {
    Button bButton = new Button("Pause/Play");
    //...[style and add bButton to the form]
    show();
    bButton.addActionListener(this);
    bgSound = new BGSound("alarm.wav");
    bgSound.play();
  }

  public void actionPerformed(ActionEvent evt) {
    bPause = !bPause;
    if (bPause)
      bgSound.pause();
    else
      bgSound.play();
  }
}
`
```


Troubleshooting Sounds

- When you start the program, if it says: “Adding CEF to classpath” on the console and then gives errors when you play sounds:

  Rename the directory called “cef” located under “C:\Users\<username>\.codenameone” to something else (e.g., rename it as “cef-backup”)

  Then when the program starts, it should not say “Adding CEF to classpath” anymore and CN1 would use JavaFX to play sounds (instead of using CEF). See Appendix-2 in Assignment #3 write-up for instructions on how to install and use JavaFX.

- After you apply the above-mentioned fix, if you are having errors when playing sounds, try setting the initial *Sound* game state value to OFF (i.e., initially assign the value of the *Sound* flag to false and start the game with sounds disabled).

Troubleshooting Sounds (cont.)

If you are still having errors after you add sounds to your assignment (e.g., you receive **`NullPointerException`** and/or your GUI does not show up properly), make sure all sounds (including **`BGSound`**) are created inside **`createSounds()`** and try having the following structure at the end of your **`Game`** constructor:

```
show(); //... [query MapView's width and height]
gw.init();
gw.createSounds(); //call createSounds() after calling init() in GameWorld
revalidate(); //call revalidate on the form to fix the GUI
//... [create UITimer and schedule it after calling createSounds()/revalidate()] ...
```

Then, have the following structure in **`Sound`** constructor (and add the same lines to **`BGSound`** constructor):

```
while(m == null) { //ADD THIS
 try {
   InputStream is = Display.getInstance().getResourceAsStream(getClass(),
                                                              "/" + filename);
   m = MediaManager.createMedia(is, "audio/wav");
 } catch (Exception e) {
   e.printStackTrace(); }
}//ADD THIS
```

Still having problems? It is possible that you do not have properly formatted wav file. To see whether this is the case, try using the following wav file used in the demo code:

`https://athena.ecs.csus.edu/~pmuyan/downloads/alarm.wav`

If all fails, reboot your system and re-try.