**#** JavaSound is the collection of classes and interfaces added to Java starting with version 1.3.

**#** Javasound is split into two parts- **MIDI and Sampled.**

**# MIDI ==**Musical Instrument Digital Interface, and is the standard protocol forgetting different kinds of electronic sounds to communicate.

**#** Synthesizer is the software only instrument built in Java to play sounds Using instructions from MIDI.

**#** To use the sequencer, import first using - **import javax.sound.midi,\*;**

#Basic code for importing sequencer -

**import javax.sound.midi.\*;**

**public class MusicTest1 {**

**public void play() {**

**Sequencer sequencer = MidiSystem.getSequencer(); System.out.println(“We got a sequencer”);**

**} // close play**

**public static void main(String[] args) {**

**MusicTest1 mt = new MusicTest1();**

**mt.play();**

}

}

**##Line 4 note** - We need a Sequencer object. It’s the main part of the MIDI device/instrument we’re using. It’s the thing that, well, sequences all the MIDI information into a ‘song’. But we don’t make a brand new one ourselves -- we have to ask the MidiSystem to give us one.

#note - the code above won’t work as we must use an exception.

**#Methods in Java use exceptions to tell the calling code,**

**“Some thing Bad Happened. I failed.” (Same as Python)**

**#** If a method throws an exception, We can find “throws “ cause in the risky method’s declaration.

**#** If you wrap the code in the **try: catch**: block, the compiler will use it to prevent outbreaks when our code fails. It is a great way to find faults in our code too. (Same as **Python’s try:except** blocks)

**#**Using this code instead of the previous sequencer getting code will work due to **try:except** block.

**Code -**

import javax.sound.midi.\*;

public class MusicTest1 {

public void play() {

**try {**

Sequencer sequencer= MidiSystem.getSequencer(); System.out.println(“Successfully got sequencer”);

} **catch(MidiUnavailableException ex) {**

**System.out.println(“Bummer”);**

}

} // close play

public static void main(String[] args) {

MusicTest1 mt = new MusicTest1();

mt.play();

}

}

**#An exception is an object...of type Exception.**

**#ex.printStackTrace(); lets us trace our error and exceptions, just like python’s sys module’s function.**

**#**Exceptions that are NOT subclasses of RuntimeException are checked for by the compiler. They’re called “checked exceptions”

**#** if you throw an exception in your code, you must declare it using **“throws**” keyword in our method declaration.

**#** if you call a function which throws an exception/ has declared that it throws an exception, you must acknowledge it and use necessary ways to get around it. One way is to use **Try:Catch** block.

# The reason why compiler lets RuntimeExceptions pass and not the other exceptions is because runtime errors are caused by logic, like a ZeroError. While the other exceptions happen because of errors not dependable on the code or code syntax error.

**#**A **try:catch** is for handing exceptions situations, not flaws in our code.

**#**If you’re not prepared to handle the exception, you can still make the compiler happy by officially ‘ducking’ the exception. We’ll talk about ducking a little later in this chapter.

#A **finally** block is where you put code that must run regardless of an exception.

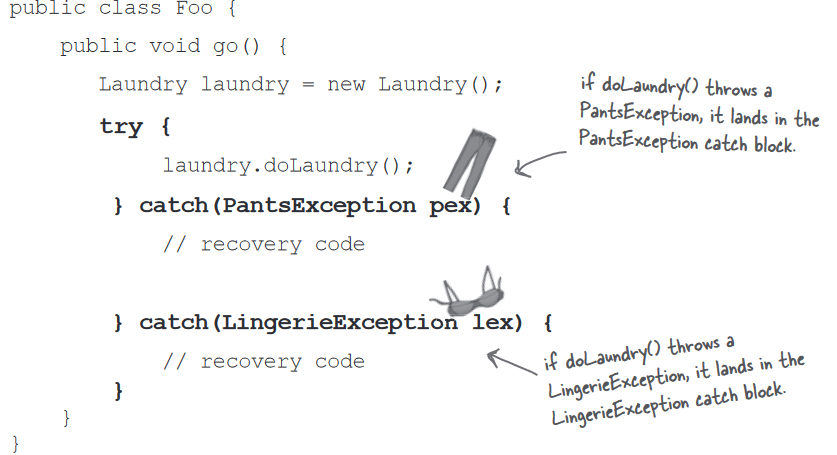
# We can get our methods to throw more than one exceptions, simply stack up the exception names after method name.

**Example-**

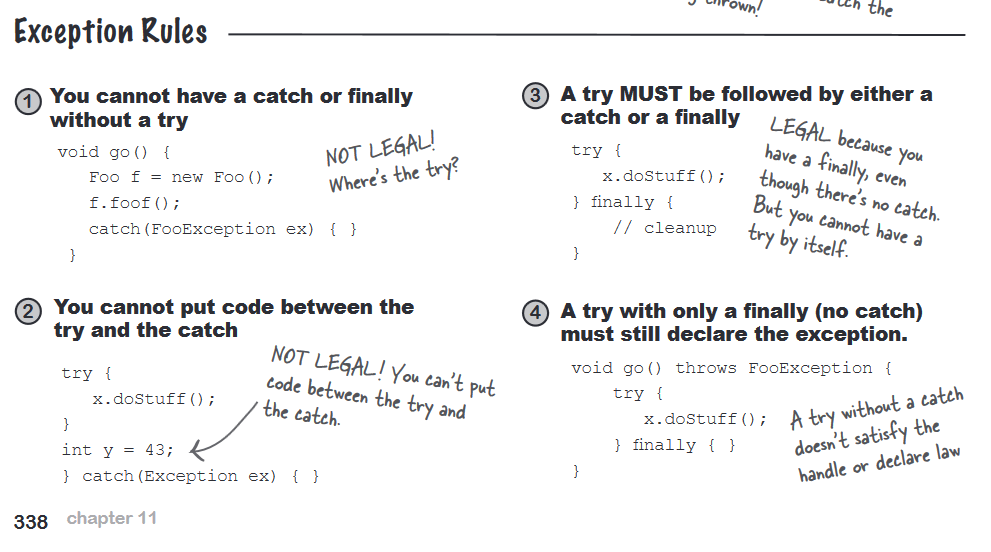
public class Laundry {

public void doLaundry() throws **PantsException, LingerieException** {

// code that could throw either exception



**#** You can DECLARE exceptions using a supertype of the exceptions you throw. You can also CATCH exceptions using a supertype of the exceptions.

**#**If you don’t want to handle an exception, you can DUCKING it by declaring it.

**#**