Review: Definitions

- **Class**: definition or template of an object
  - `class Book`  
- **Object**: specific instance of a class
  - `Book book1;`
- **Method**: action that a class performs
  - `book1.turnPage(32);`
- **Variable**: data used by a class (also called `data member`)
  - `book1.title = "War and Peace";`

Deconstructing `MusicFile.java`

- Class starts with `comments`
- Then the **class declaration**
  - `class MediaFile`
  - gives the name of the class, and also its “visibility” (more on this later)
- Then come the **variable declarations**
- Then the **methods**

Variables

- All variables have three parts:
  - `type`: integer, real number, string, etc.
  - `name`: its label
  - `value`: what is assigned to that variable
- **Example**: `String myName = "Amy";`
  - `type = String`
  - `name = myName`
  - `value = "Amy"`
**Variable types**

- Specify:
  - what operations can be done on that variable
    - e.g., can divide two numbers but not two strings; can concatenate two strings but not two numbers
  - how much space Java needs to set aside to store the variable
    - int (integer): 32 bits
    - double (real number): 64 bits
    - String: varies (Java sets aside 32 bits to store the “address” of where the variable will actually be stored)

**Methods**

- “Actions” that an object takes
- 3 parts:
  - return type: what type of value this method will send back to whomever called it
    - void setGenre(String g): returns nothing, so return type is void
    - int addTwo(int a, int b): returns the sum of two integers, so return type is int
  - name
  - list of parameters: data values that the method needs in order to take its action
    - in examples above: parameters are g, a, b

**Comments**

- Statements that are not executed when the program is run
- “Notes” to the programmer, describing what the program (or various parts of it) does
- Two styles:
  - //: everything between // and the end of the line is treated as a comment (single line only)
  - /* ... */: everything between /* and */ is treated as a comment (can be multiple lines)

**Two special types of methods**

- main(): necessary for every program that we would like to execute
  - can have programs that just define a class (MusicFile.java, e.g.) and programs that just use objects of other classes (MediaCenter.java, e.g.)
  - the latter type should contain a main() method
- constructor: method that initializes or “sets up” an object of a class by setting variable values, etc.
Constructor

- looks like a method
  - has a name
  - typically has a list of parameters (input values)
  - but has no return type because it does not return anything (ever)
- contains the basic set of instructions to make an object usable
  - example: for MusicFile objects, we need to at the very least know the artist and the title of a song

main() method

- always starts with public static void main(String[] args)
  - public: everyone can see and use this method
  - static: more on this in a minute
  - void: returns nothing
  - main: name of the method
  - String[] args: input parameters (this must always be included, even though main has a different idea about input parameters....more on this later)

More on methods

- We can't use a method or access a data member of a class until we've instantiated an object of that class. Only then can we access these, and we access them through the object itself.
- Q: What about these statements from Friday's lab, then?
  - JOptionPane.showMessageDialog(null,"...");
  - name1 = JOptionPane.showInputDialog(null,"...");
  - System.out.println("...");

Static methods

- A particular type of method that can be used without having to instantiate an object of its class
- Denoted by the static keyword
- Some commonly-used methods in “miscellaneous” classes are static (the ones on the previous page, for example, and some mathematical functions)
- We'll use static methods, but we won't write them in this class
- Use a static method by typing [class name].[method name](...)
**main() is a static method**

- But we call it differently:
  - typing `java [class name]` is equivalent to saying `[class name].main(...)` (but the latter will not work)

**Instantiating an object**

- means we “create” an object of a class
- means that Java sets aside a certain amount of memory to store an object of that class (based on what data it needs to store, what methods it has, etc.)
- means the constructor for that class is called

**Example**

- **example:** `MusicFile music1 = new MusicFile("Benny and the Jets", "Elton John");`
  - type (class) = `MusicFile`
  - name = `music1`
  - `new` means “call the constructor”
  - “Benny and the Jets” and “Elton John” are the parameters we’ll pass in to the constructor (needed in order to initialize and use the class)