CSE 115 / 503
INTRODUCTION TO COMPUTER SCIENCE I

Dr. Carl Alphonce
Dr. Jesse Hartloff

University at Buffalo
School of Engineering and Applied Sciences

CSE 50
1967-2017

10/16/17 Announcements

Snapshot of TopHat and Friday Activity grades added to AutoLab gradebook

Some changes coming (more details to come)

- **recitations – increased TA interaction time**
  - **even weeks – quizzes as activity**
  - **odd weeks – coding exercises**
- **review / extra help sessions early in week (for previous week's material)**
- **added opportunities to earn proficiency points**

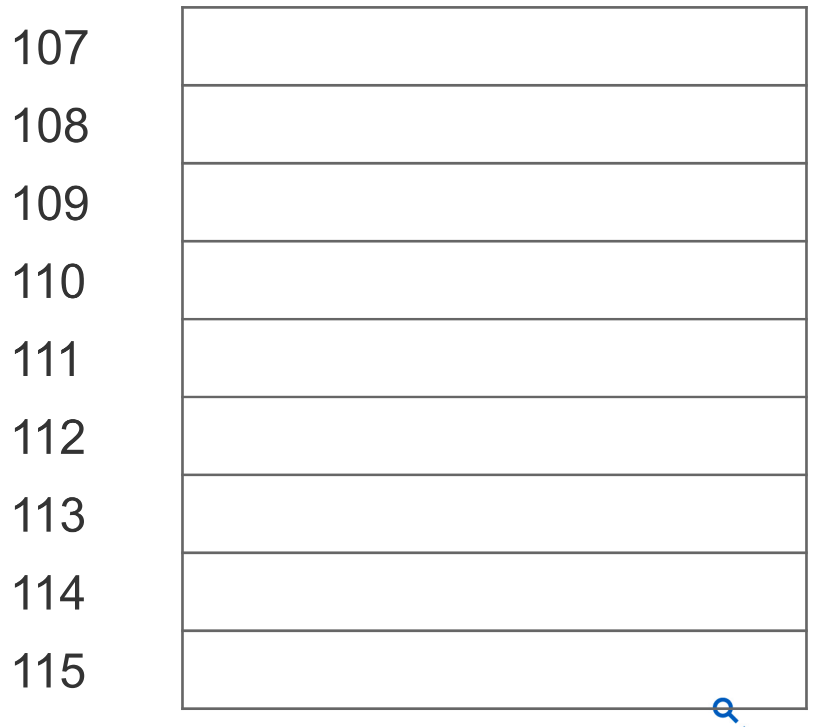
memory

The memory of a computer is arranged into locations called bytes. Each byte has a unique address. Addresses start at 0 and increment by one for each subsequent byte.

For example, part of memory can be depicted as in the diagram at right.

Each rectangle in the diagram represents a byte. The address of each type is shown immediately to its left.

The bytes in this diagram have addresses from 107 to 115.



memory

At any given point in time some locations in memory will be used (holding data we care about) and some will be available (holding something, but not data we care about).

The diagram at right shows that the byte at address 107 is used, while the bytes from address 108 to address 115 are available for use.

107	used
108	available
109	available
110	available
111	available
112	available
113	available
114	available
115	available

Consider this block of code:

```
int x;
x = 17;
```

What happens in memory when this code is executed?

First, memory for the variable is set aside (let's say the memory at address 108)…

107	used
108	reserved
109	available
110	available
111	available
112	available
113	available
114	available
115	available

Consider this block of code:

```
int x;
x = 17;
```

What happens in memory when this code is executed?

First, memory for the variable is set aside (let's say the memory at address 108), and then a representation of the value 17 is stored in that location.

107	used
108	17
109	available
110	available
111	available
112	available
113	available
114	available
115	available

Consider this block of code:

```
int x;
x = 17;
```

What happens in memory when this code is executed?

First, memory for the variable is set aside (let's say the memory at address 108), and then a representation of the value 17 is stored in that location.

(The value 17 is actually stored as a sequence of zeroes and ones using an encoding called two's complement.)

107	used
108	00010001
109	available
110	available
111	available
112	available
113	available
114	available
115	available

OO software systems are systems of interacting objects.

Objects have

properties:
- these are things that objects know
- e.g. what you had for breakfast

behaviors:
- these are things objects do
- e.g. being able to reply to the question “What did you have for breakfast?”

Evaluating

```
new code.ratables.Song("sdTSUkIJxnU","Offenbach","Georgia on my Mind")
```

produces *a value* (which we call a reference)

causes a *side effect* (an object is created and initialized)

We can remember a reference value by storing it in a variable

When evaluating an expression like

```
new code.ratables.Song("sdTSUkIJxnU","Offenbach","Georgia on my Mind")
```

the operator ‘new’ first determines the size of the object to be created (let us say it is four bytes for the sake of this example).

Next, new must secure a contiguous block of memory four bytes large, to store the representation of the object.

Bit strings representing the object are written into the reserved memory locations. In this example we use “10101010” to indicate that some bit string was written into a given memory location; the exact bit string written depends on the specific details of the object.

The **starting address** of the block of memory holding the object’s representation is the value of the ‘new’ expression. This address is called a ‘***reference***’.

4703	used
4704	10101010
4705	10101010
4706	10101010
4707	10101010
4708	available
4709	available
4710	available
4711	used

Let us now consider an extended example.

What happens in memory?

```
Song s1;
Song s2;
s1 = new Song("RPQD5RT_MPg","Jay-Z & Alicia Keys","Empire State of Mind");
s2 = new Song("sdTSUkIJxnU","Offenbach","Georgia on my Mind");
```

Something like this!

```
Song s1;
Song s2;
s1 = new Song("RPQD5RT_MPg","Jay-Z & Alicia Keys","Empire State of Mind");
s2 = new Song("sdTSUkIJxnU","Offenbach","Georgia on my Mind");
```

12203	used	497362	available
12204	available	497363	available
12205	available	497364	available
12206	available	497365	available
12207	available	497366	available
12208	available	497367	available
12209	available	497368	available
12210	available	497369	available
12211	available	497370	used

```
Song s1;
Song s2;
s1 = new Song("RPQD5RT_MPg","Jay-Z & Alicia Keys","Empire State of Mind");
s2 = new Song("sdTSUkIJxnU","Offenbach","Georgia on my Mind");
```

Each declaration is used by the compiler to ensure that memory is set aside for the corresponding variable

12203	used	497362	available
12204	available	497363	available
12205	available	497364	available
12206	*reserved for variable s1*	497365	available
12207	available	497366	available
12208	available	497367	available
12209	available	497368	available
12210	available	497369	available
12211	available	497370	used

Both s1 and s2 have space reserved

```
Song s1;
Song s2;
s1 = new Song("RPQD5RT_MPg","Jay-Z & Alicia Keys","Empire State of Mind");
s2 = new Song("sdTSUkIJxnU","Offenbach","Georgia on my Mind");
```

12203	used	497362	available
12204	available	497363	available
12205	available	497364	available
12206	*reserved for variable s1*	497365	available
12207	*reserved for variable s2*	497366	available
12208	available	497367	available
12209	available	497368	available
12210	available	497369	available
12211	available	497370	used

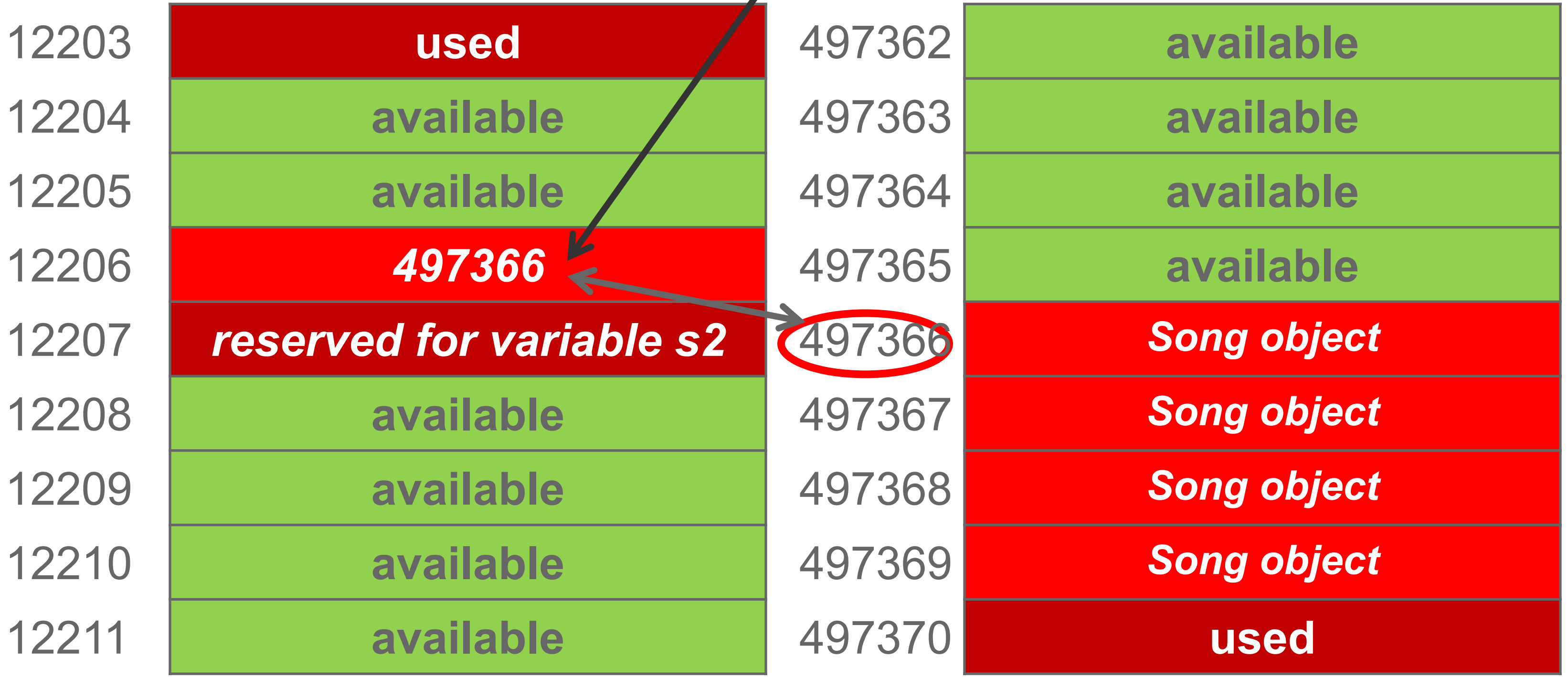
```
Song s1;
Song s2;
s1 = new Song("RPQD5RT_MPg","Jay-Z & Alicia Keys","Empire State of Mind");
s2 = new Song("sdTSUkIJxnU","Offenbach","Georgia on my Mind");
```

The value of the *new* expression is the starting address of the block of memory holding the representation of the newly created object.

12203	used	497362	available
12204	available	497363	available
12205	available	497364	available
12206	***reserved for variable s1***	497365	available
12207	***reserved for variable s2***	497366	***Song object***
12208	available	497367	***Song object***
12209	available	497368	***Song object***
12210	available	497369	***Song object***
12211	available	497370	used

The assignment stores that value in the space set aside for the variable *s1*

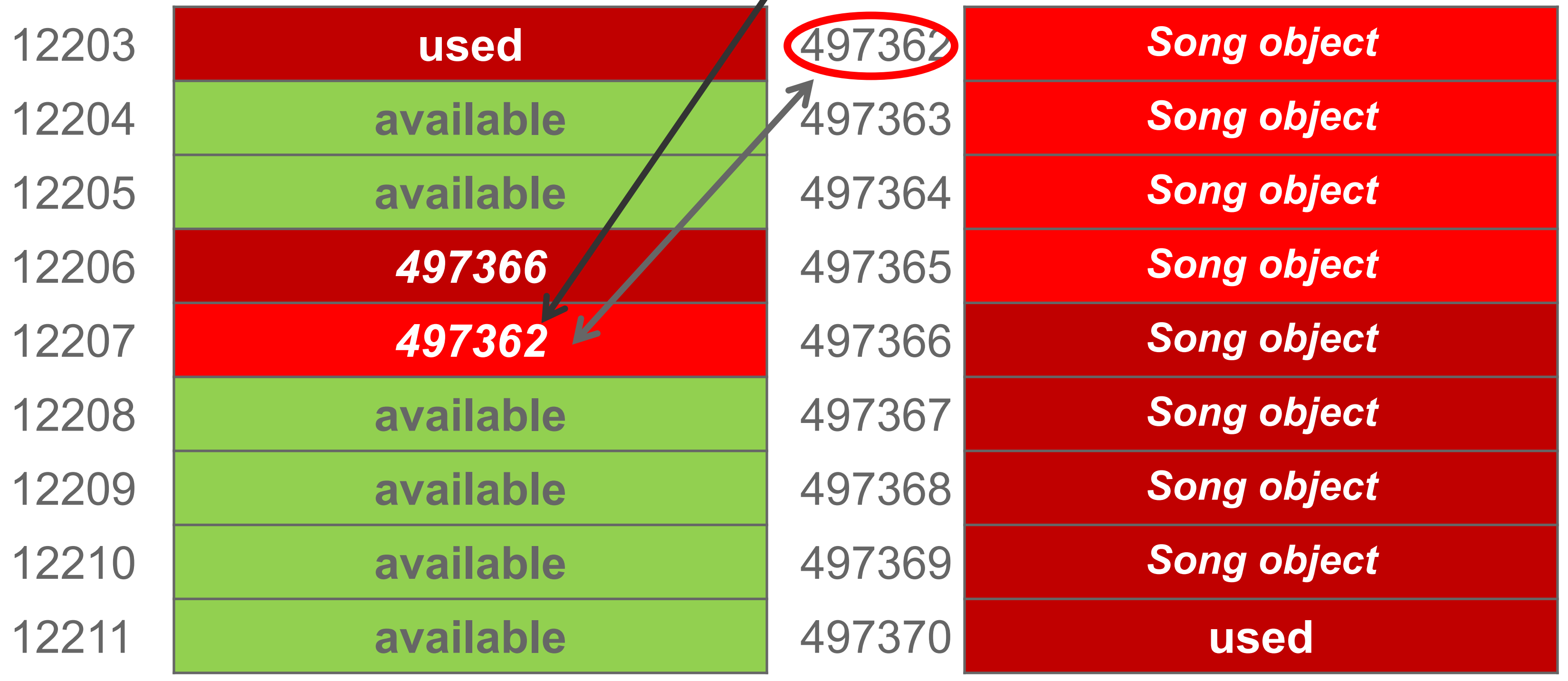
```
Song s1;
Song s2;
s1 = new Song("RPQD5RT_MPg","Jay-Z & Alicia Keys","Empire State of Mind");
s2 = new Song("sdTSUkIJxnU","Offenbach","Georgia on my Mind");
```



12203	used	497362	available
12204	available	497363	available
12205	available	497364	available
12206	*497366*	497365	available
12207	*reserved for variable s2*	497366	*Song object*
12208	available	497367	*Song object*
12209	available	497368	*Song object*
12210	available	497369	*Song object*
12211	available	497370	used

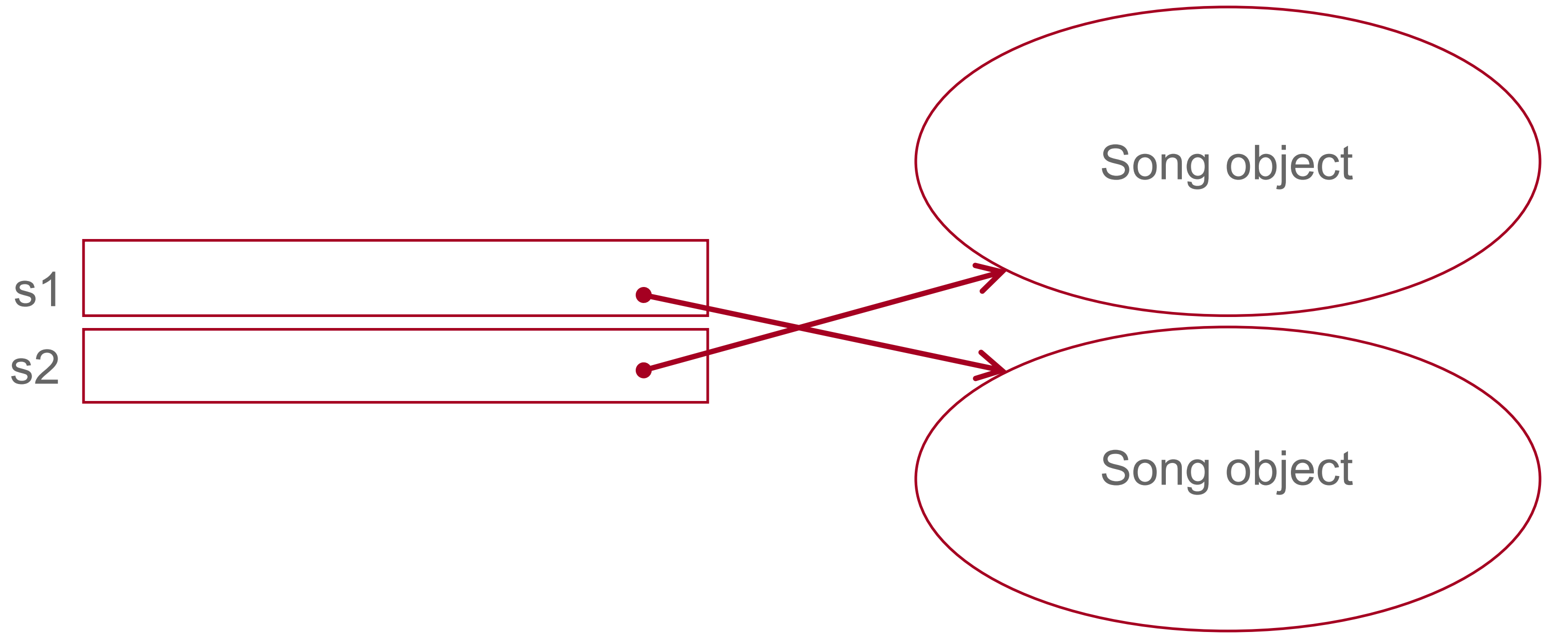
The same applies to the creation of the second new Song object, and the assignment of its reference to the variable *s2*

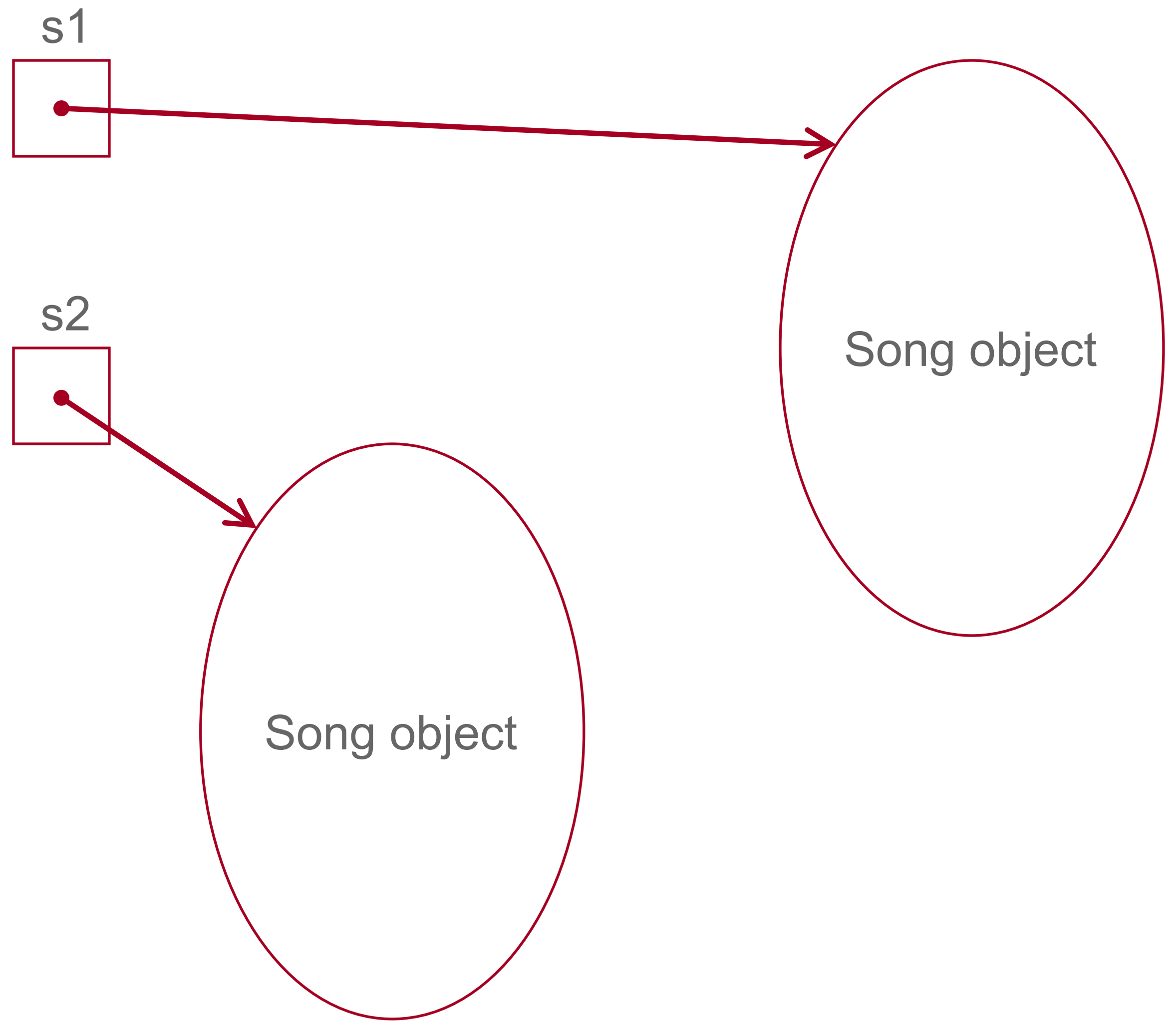
```
Song s1;
Song s2;
s1 = new Song("RPQD5RT_MPg","Jay-Z & Alicia Keys","Empire State of Mind");
s2 = new Song("sdTSUkIJxnU","Offenbach","Georgia on my Mind");
```



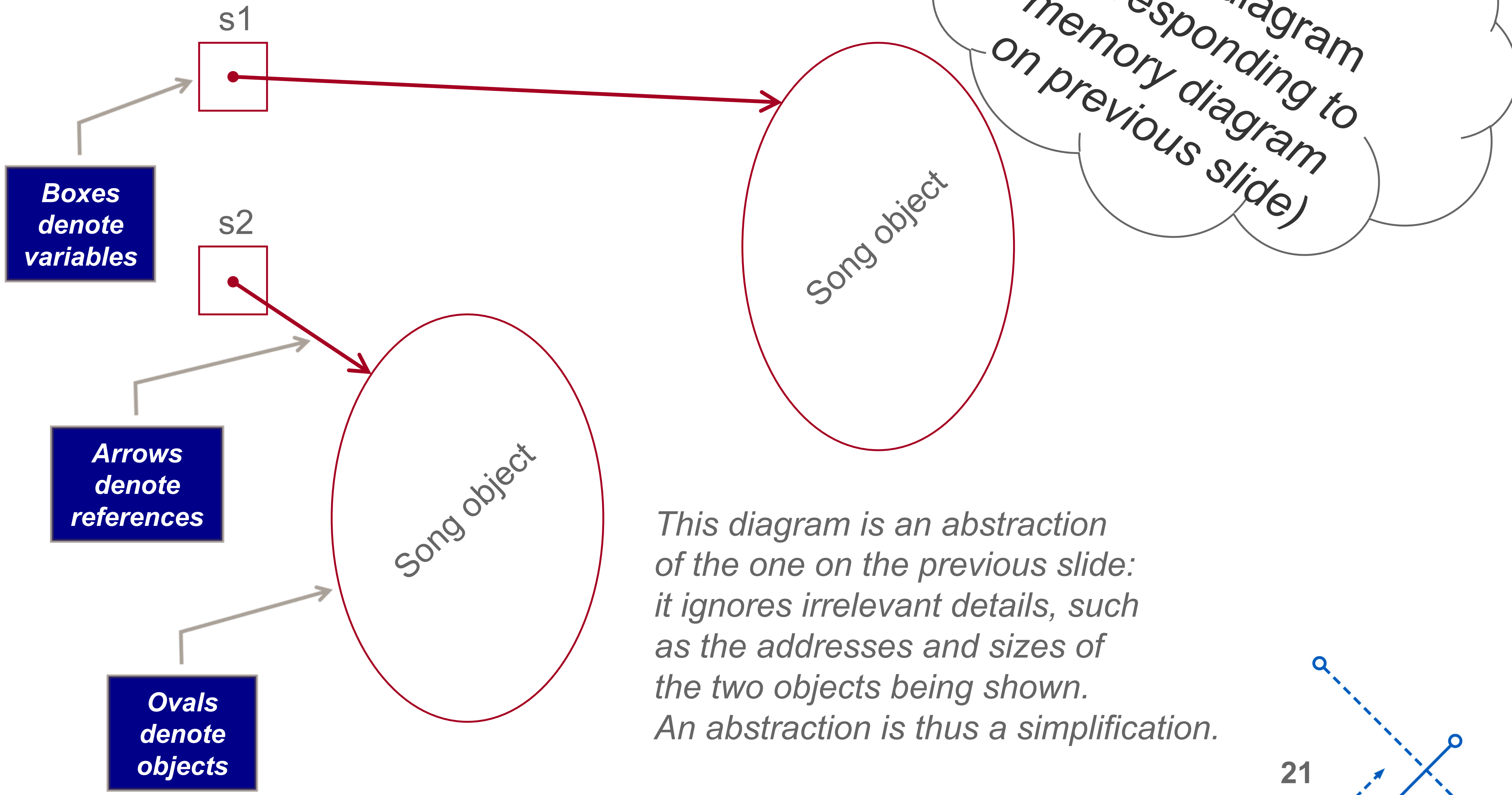
12203	used	497362	*Song object*
12204	available	497363	*Song object*
12205	available	497364	*Song object*
12206	*497366*	497365	*Song object*
12207	*497362*	497366	*Song object*
12208	available	497367	*Song object*
12209	available	497368	*Song object*
12210	available	497369	*Song object*
12211	available	497370	used

12203	used	497362	*Song object*
12204	available	497363	*Song object*
12205	available	497364	*Song object*
12206	*497366*	497365	*Song object*
12207	*497362*	497366	*Song object*
12208	available	497367	*Song object*
12209	available	497368	*Song object*
12210	available	497369	*Song object*
12211	available	497370	used


Song object
s1
s2
Song object



We can rearrange the diagram to improve readability



This diagram is an abstraction of the one on the previous slide: it ignores irrelevant details, such as the addresses and sizes of the two objects being shown. An abstraction is thus a simplification.

TopHat!

Let's take the example a little further

```
Song s2;
s2 = new Song("sdTSUkIJxnU","Offenbach",
              "Georgia on my Mind");
```

12203	used
12204	available
12205	available
12206	available
12207	available
12208	available
12209	available
12210	available
12211	available

497362	availabl
497363	available
497364	available
497365	available
497366	available
497367	available
497368	available
497369	available
497370	available
497371	available
497372	available
497373	available
497374	available
497375	available
497376	available

```
Song s2;
s2 = new Song("sdTSUkIJxnU","Offenbach",
              "Georgia on my Mind");
```

12203	used
12204	available
12205	available
12206	available
12207	available
12208	available
12209	available
12210	available
12211	available

497362	available
497363	availab
497364	availab
497365	avai
497366	avai
497367	avail
497368	availa
497369	available
497370	"sdTSUkIJxnU"
497371	"Georgia on my Mind"
497372	available
497373	"Offenbach"
497374	available
497375	available
497376	available

The argument Strings exist in memory too!

```
public class Song{


    private ArrayList<Integer> ratings;
    private String youtubeID;
    private String artist;
    private String title;

    .
    .
    .

}
```

This is how the Song class is defined

```
public class Song{

    private ArrayList<Integer> ratings;
    private String youtubeID;
    private String artist;
    private String title;

    .
    .
    .

}
```

This is how the Song class is defined

The instance variables make up the representation of the object in memory.

Each of these variables is allocated space inside the object's memory block.

In this example, each variable is of a reference type, so each variable will hold a reference to another object.

```
Song s2;
s2 = new Song("sdTSUkIJxnU","Offenbach",
              "Georgia on my Mind");
```

12203	used	
12204	497362	s2
12205	available	
12206	available	
12207	available	
12208	available	
12209	available	
12210	available	
12211	available	

497362	reserved	ratings
497363	reserved	youtubeID
497364	reserved	artist
497365	reserved	title
497366	available	
497367	available	
497368	available	
497369	available	
497370	"sdTSUkIJxnU"	
497371	"Georgia on my Mind"	
497372	available	
497373	"Offenbach"	
497374	available	
497375	available	
497376	available	

What does the constructor do?

```
public class Song {

    private ArrayList<Integer> ratings;
    private String youtubeID;
    private String artist;
    private String title;

    .
    .
    .
    public Song(String youtubeID, String artist, String title) {
        this.youtubeID = youtubeID;
        this.artist = artist;
        this.title = title;
        this.ratings = new ArrayList<>();
    }

}
```

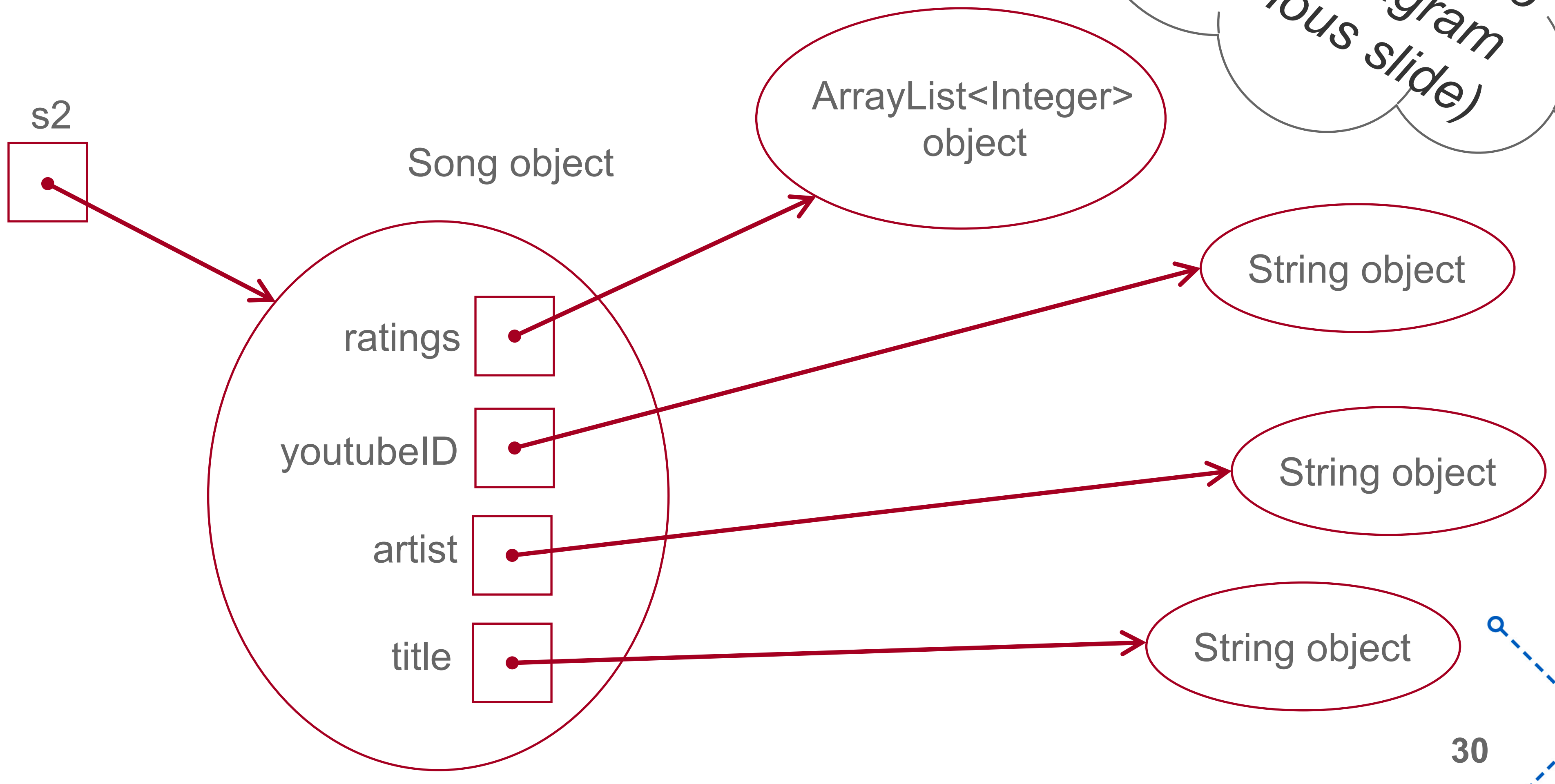
The constructor initializes the variables to sensible initial values.

```
Song s2;
s2 = new Song("sdTSUkIJxnU","Offenbach",
              "Georgia on my Mind");
```

12203	used	
12204	497362	s2
12205	available	
12206	available	
12207	available	
12208	available	
12209	available	
12210	available	
12211	available	

497362	497375	ratings
497363	497370	youtubeID
497364	497373	artist
497365	497371	title
497366	available	
497367	available	
497368	available	
497369	available	
497370	"sdTSUkIJxnU"	
497371	"Georgia on my Mind"	
497372	available	
497373	"Offenbach"	
497374	available	
497375	ArrayList<Integer>	
497376	available	

Simpler! Object diagram (corresponding to memory diagram on previous slide)



Questions?