Some background before I start the lesson...

- Initial lessons use existing classes (String, Arraylist, Random)
  - practice reading APIs
  - practice instantiating objects
  - practice invoking static and instance methods
- Today's Lesson: Defining a new Java class
  - Delay introduction of constructors
  - Initial emphasize on object state and object references
  - Visual debugger to clarify object concepts, avoid common misconceptions

Review : Java is an Object-Oriented Language

Object	State (properties)	Behavior (access & modify state)
Mobile Phone	brand model is on volume	toggle on/off adjust volume send text
Random	seed	nextInt nextBoolean, ...
ArrayList	list elements	add element delete element

Review : Java Data Types

Java Data Types		
Primitive Types	byte, short, int, long, float, double, boolean, char	Variable stores a primitive value
Reference Types (non-primitive)	String, ArrayList, Random, JButton, JFrame, ...	Variable stores an object reference

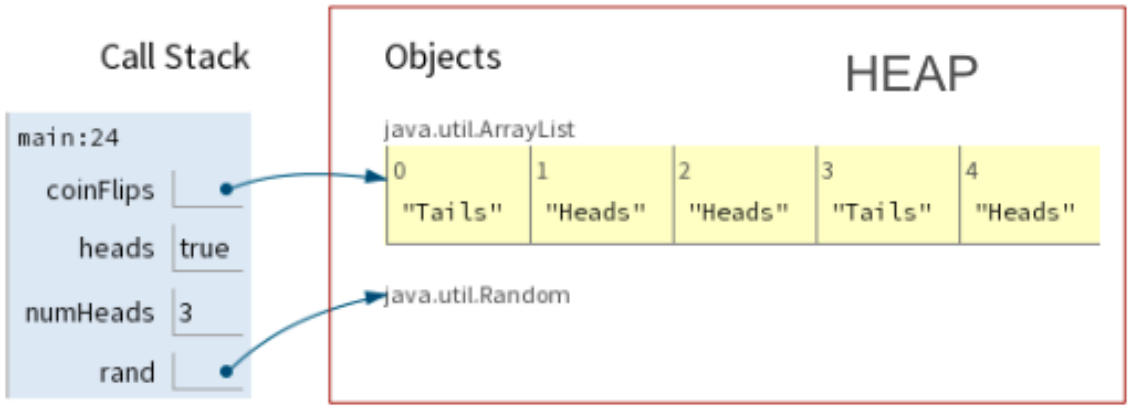
- Primitive types are predefined in Java.
- Reference types can be defined by the programmer.

Review : Java Data Types

```
public static void main(String[] args) {
  ArrayList<String> coinFlips = new ArrayList<String>();
  Random rand = new Random();
  int numHeads = 0;
  boolean heads = rand.nextBoolean();

  while (numHeads < 3) {
    if (heads) {
      numHeads++;
      coinFlips.add("Heads");
    } else {
      coinFlips.add("Tails");
    }
    heads = rand.nextBoolean();
  }

  System.out.println("Total coin flips:" + coinFlips.size());
  System.out.println(coinFlips);
}
```



Today's Lesson - Defining a new Java class

- We've seen how to use existing Java classes (String, ArrayList, etc.) to solve some interesting problems
- Today we'll see how to define **new** classes to model some real world objects such as common types of pets (Fish, Cat, Hamster, Dog)
- A Java class is a blueprint for describing similar objects
    - fields describe object state
    - methods implement object behavior

```
public class ClassName {

  //Field declarations

  //Method declarations

}
```

A class to model pet fish

```
public class Fish {

    //Field declarations
    int age;
    boolean isAggressive;
    String species;

}
```

Objects

Fish instance

age	15
isAggressive	false
species	"Goldfish"

Fish instance

age	8
isAggressive	true
species	"Red Tail Shark"

Creating a new class instance (i.e. object)

<table> <tr> <td>

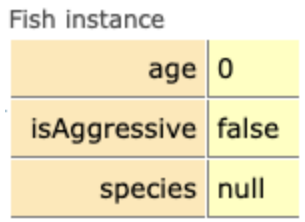
```
public class Fish {
    int age;
    boolean isAggressive;
    String species;
}
```

</td> <td>

blah

</td> </tr> </table>

Creating a new class instance (i.e. object)

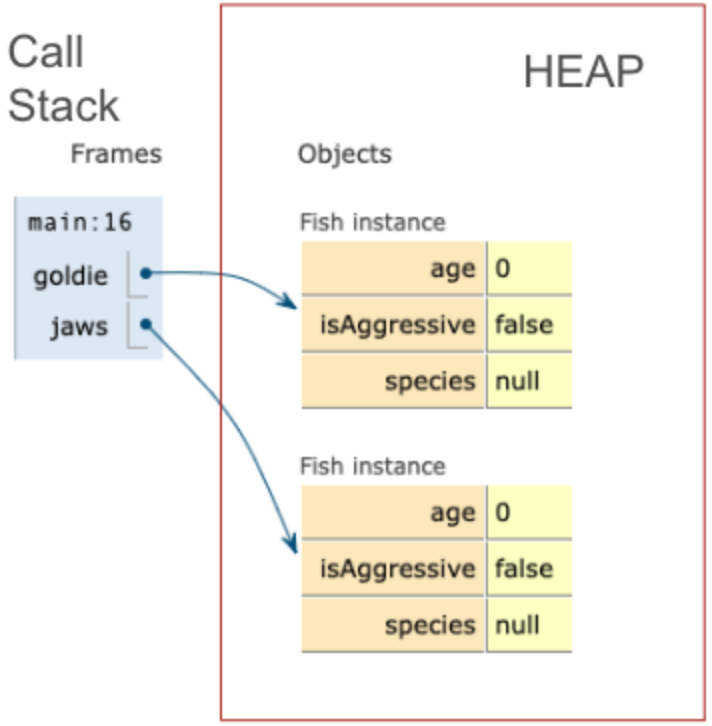
Java Expression	Heap (dynamic memory)	
`new Fish()`		1. Memory is allocated to store fields 2. Fields are initialize with default values based on data type 3. Object reference is returned

Reference Variable

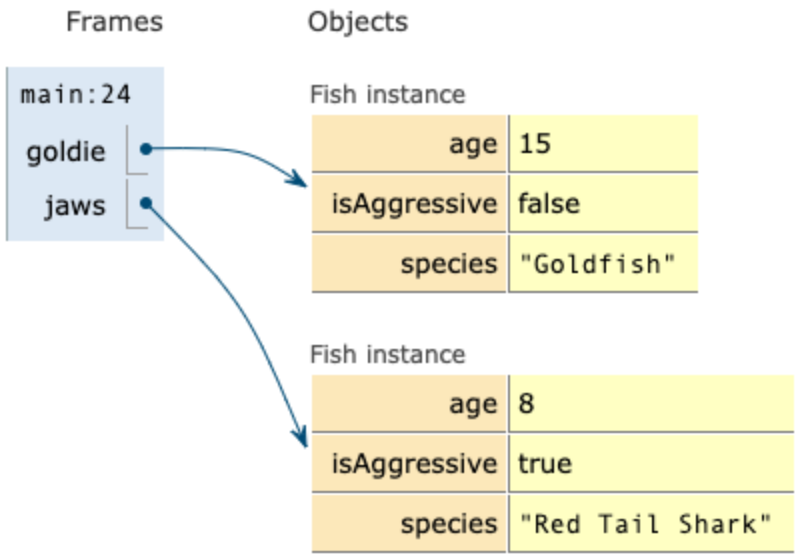
A **reference variable**:

- Is declared with a reference data type (such as class **Fish**).
- Stores an object reference or `null`.

```
Fish goldie = new Fish();
Fish jaws = new Fish();
```



Suppose we'd like to update both fish as shown:



- Each fish instance has it's own variable named **age**.
- **Dot notation** is used to access a field through a reference.

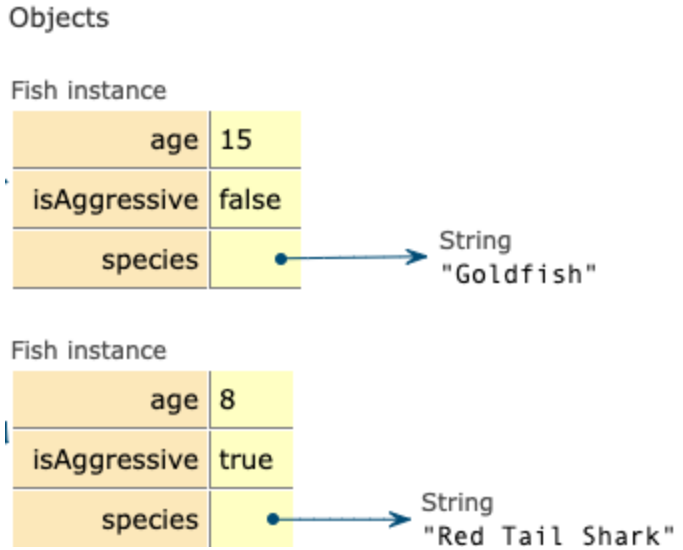
`objectReference.fieldName`

```
goldie.age = 15;
goldie.species = "Goldfish";

jaws.age = 8;
jaws.species= "Red Tail Shark";
jaws.isAggressive = true;
```

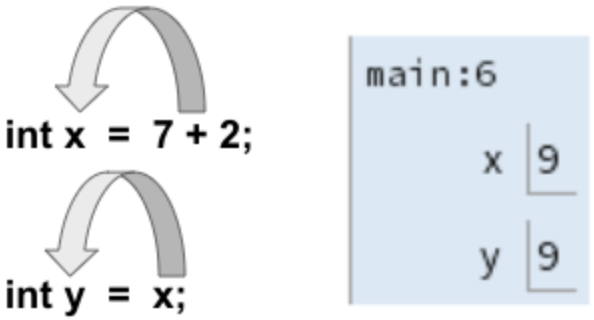
NOTE : `String` is a reference data type

The species variable actually stores a reference to a separate **String** object.

String Literal (default view)	String Reference
	

Recall how an assignment statement works

The value of the expression on the right hand side is copied into the variable on the left hand side.



```
public class Cat {

    String name;
    boolean isPurring;

    public static void main(String[] args) {
        Cat calico = new Cat();
        Cat tabby = new Cat();
        Cat favorite = calico;

        tabby.name = "Maru";
        calico.name= "Chestnut";
        favorite.isPurring = true;

        System.out.printf("calico: %s %b%n", calico.name, calico.isPurring);
        System.out.printf("tabby %s %b%n", tabby.name, tabby.isPurring);
        System.out.printf("favorite: %s %b%n", favorite.name, favorite.isPurring);
```

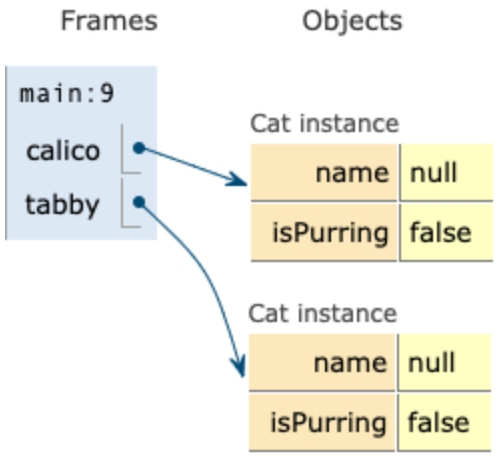
- Sketch out the heap and stack frame.
- What gets printed? Debug to confirm your answer.

`new Cat()` creates an instance

<details> <summary>

```
Cat calico = new Cat();
Cat tabby = new Cat();
```

</summary>



</details>

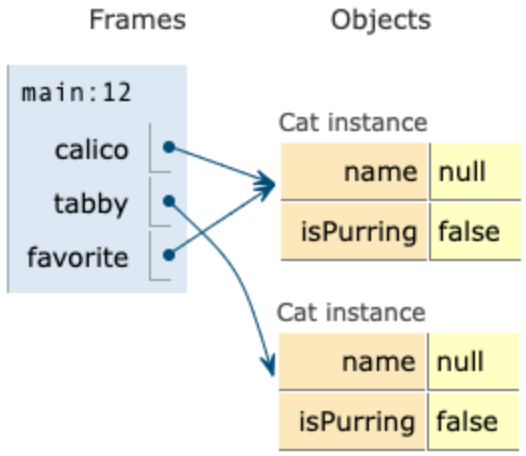
Multiple variables can reference the same object

<details> <summary>

- Two primitive variables can store the same value.
- Two reference variables can reference the same object.

```
Cat calico = new Cat();
Cat tabby = new Cat();
Cat favorite = calico
```

</summary>



</details>

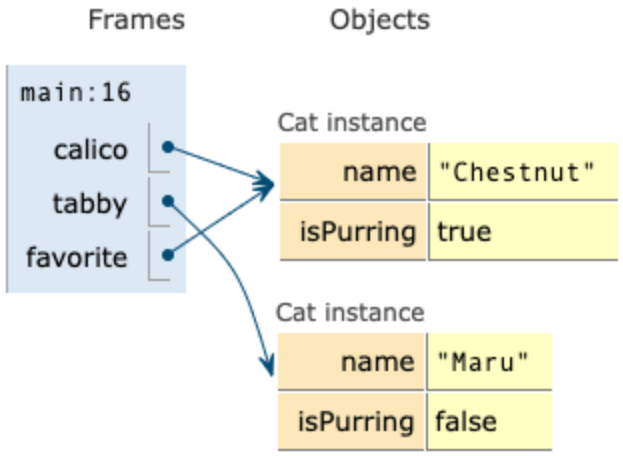
Updating object state

<details> <summary>

```
Cat calico = new Cat();
Cat tabby = new Cat();
Cat favorite = calico;

tabby.name = "Maru";
calico.name= "Chestnut";
favorite.isPurring = true;
```

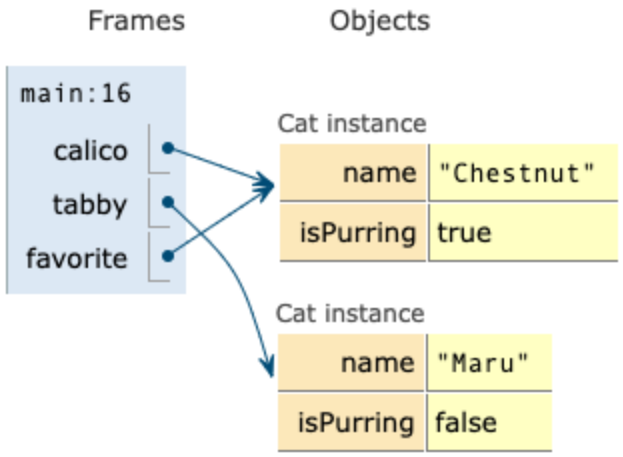
</summary>



</details>

What get's printed?

```
System.out.printf("calico: %s %b%n", calico.name, calico.isPurring);
System.out.printf("tabby %s %b%n", tabby.name, tabby.isPurring);
System.out.printf("favorite: %s %b%n", favorite.name, favorite.isPurring);
```




```
calico: Chestnut true
tabby: Maru false
favorite: Chestnut true
```

CHALLENGE

- Implement a class named `Hamster` with fields to store a name, weight in ounces, and whether they are friendly.
- Implement a `main` method to instantiate two hamster and update their state as shown.
    - do not write unnecesary field assignments (consider default initialization).
- Step through with the debugger to confirm your code is correct.