Some background before I start the lesson...

- Initial lessons use existing classes (String, Arraylist, Random, Swing)
  - Practice reading APIs
  - Practice instantiating objects
  - Practice invoking methods
  - Using Visual/IDE debugger to introduce concepts of objects, state, references
- Today's Lesson: Defining a new Java class
  - Initial emphasize on object state and object references
  - Visual debugger to clarify object concepts
    (map static syntax to dynamic
    runtime model)
  - Introduce constructors/methods **after** introducing object references

Today's Lesson - Defining a new Java class

- We've seen how to use existing Java classes (String, ArrayList, Random, etc.) to solve a few interesting problems
- Today we'll learn how to define **new** classes to model the state and behavior of some real world objects

Review: Java is an Object-Oriented Language

Object	State (properties)	Behavior (access & modify state)
Mobile Phone	model is on volume, ...	toggle on/off adjust volume send text, ...
Random Number Generator	seed, multiplier,...	generate random integer, random boolean, ...
List	list elements	add element delete element get size, ...

Defining a new Java class

A **Class** is a blueprint for describing similar objects

- fields (instance variables) describe object state
- methods implement object behavior (access and update state)

```
public class ClassName {

  //Field declarations

  //Method declarations

}
```

Defining a new Java class

```java
public class Fish {

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

}
```

An object is an **instance** of a class

Objects

Fish instance

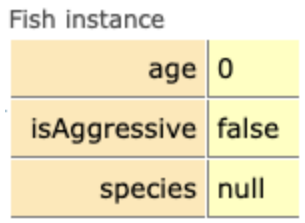
age	15
isAggressive	false
species	"Goldfish"

Fish instance

age	8
isAggressive	true
species	"Red Tail Shark"

Creating a new `Fish` instance

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

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

Quick Review: Variable declarations must specify data type

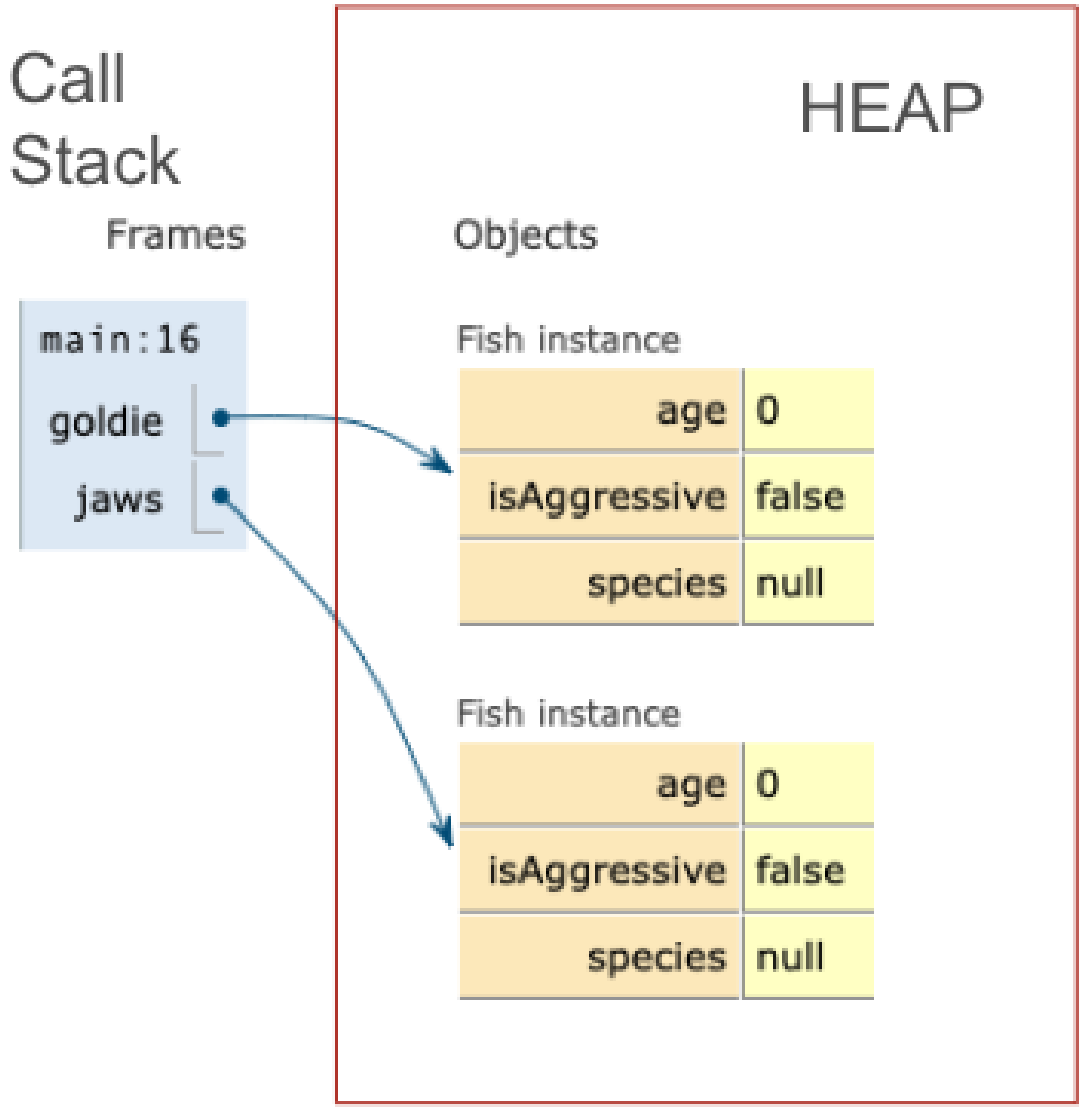
Java Data Types	Examples	Variable
Primitive Types	byte, short, int, long, float, double, boolean, char	Store primitive value (7, 3.5, true, 'a')
Reference Types (non-primitive)	String, ArrayList, Random, Fish, ...	Store object reference (info about memory location)

A class is one kind of reference type (there are others as well)

Reference Variable

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

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



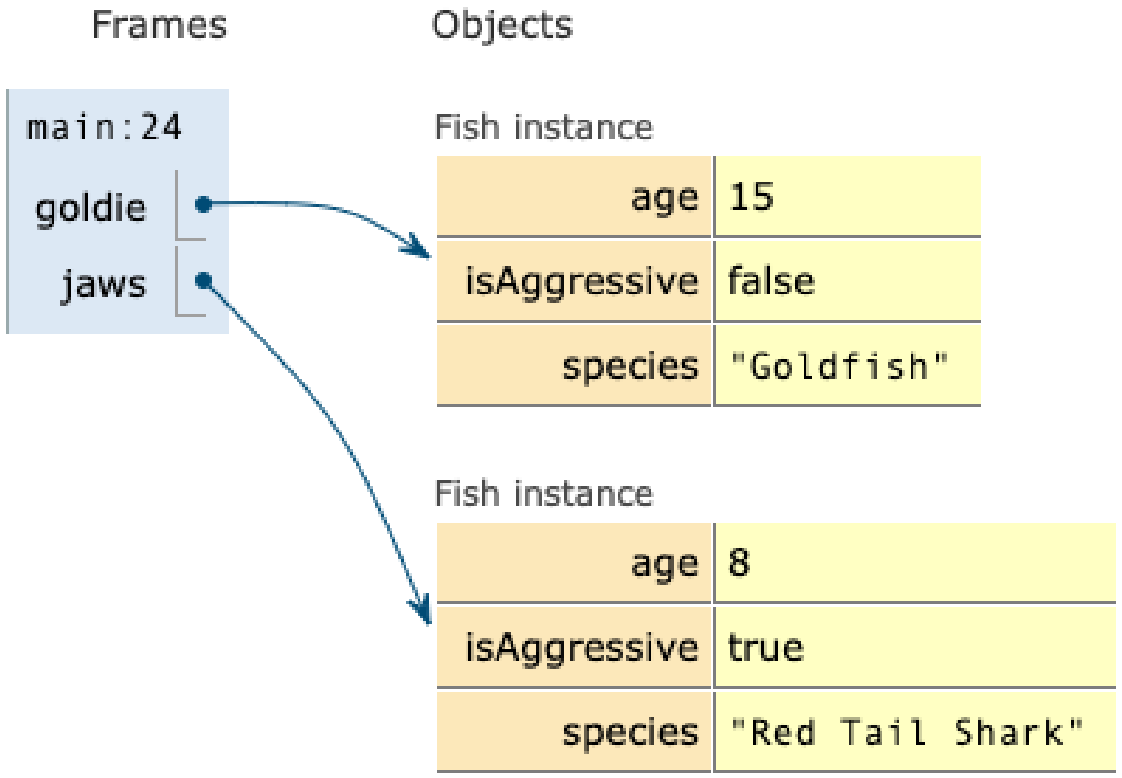
Updating object state

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

`objectReference.fieldName`

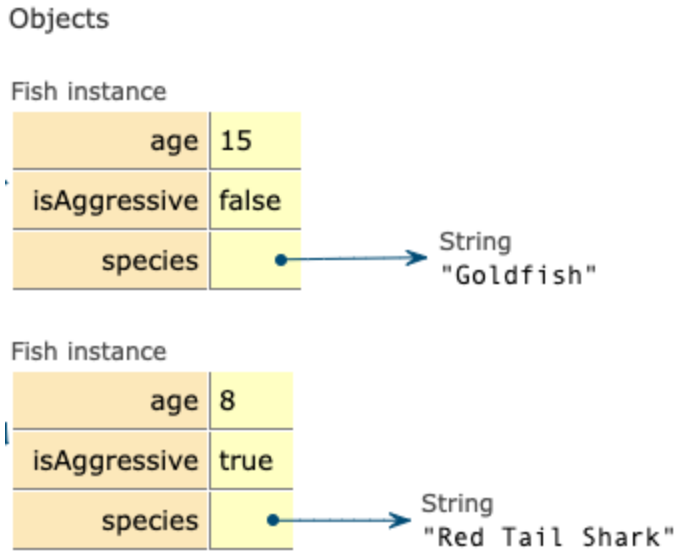
```
goldie.age = 15;
goldie.species = "Goldfish";

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

pythontutor.com visualization



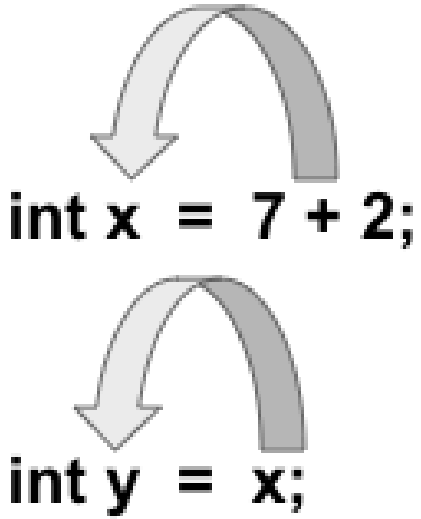
NOTE : `String` is a reference data type

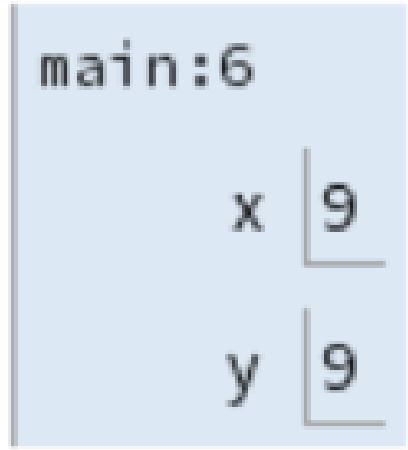
`species` actually stores an object reference

String Literal (default view)	String Reference
	

Recall how an assignment statement works

1. evaluate expression on right hand side
2. copy value into variable on left hand side



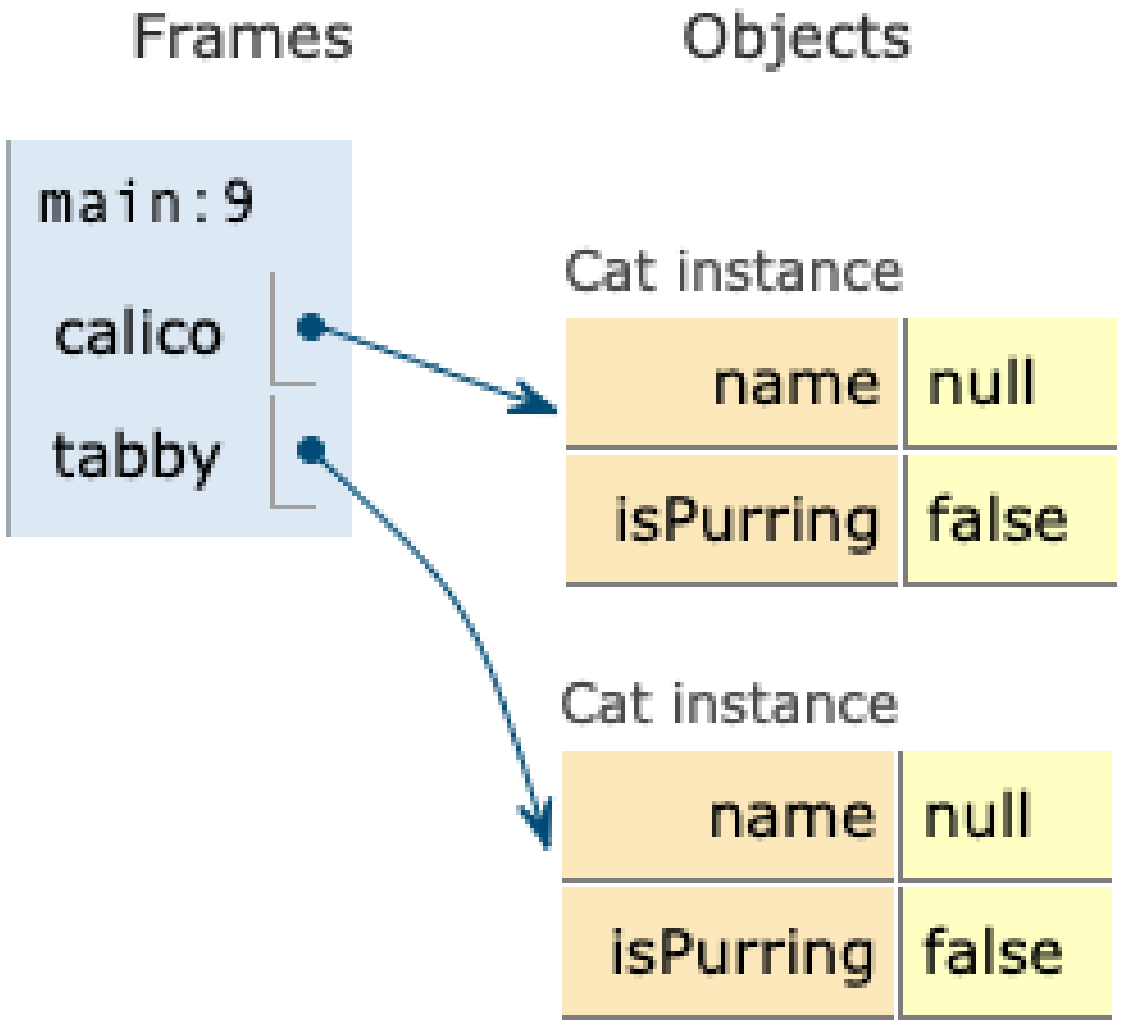


CHALLENGE

```
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.println(calico.name + "," + calico.isPurring);
    System.out.println(tabby.name + "," + tabby.isPurring);
    System.out.println(favorite.name + "," + favorite.isPurring);
  }
}
```

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

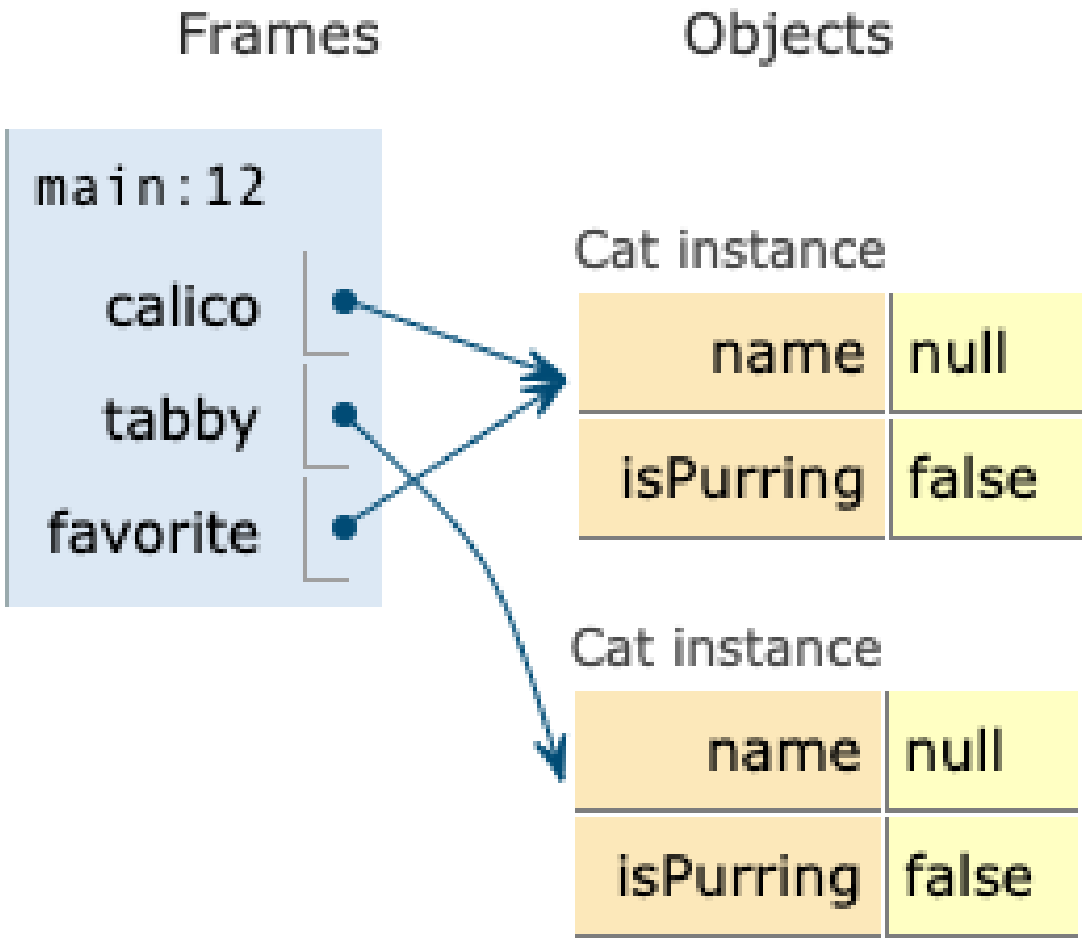
- `new Cat()` creates `Cat` instance with fields initialized to default values
- `calico` and `tabby` reference the new objects



Multiple variables can reference the same object

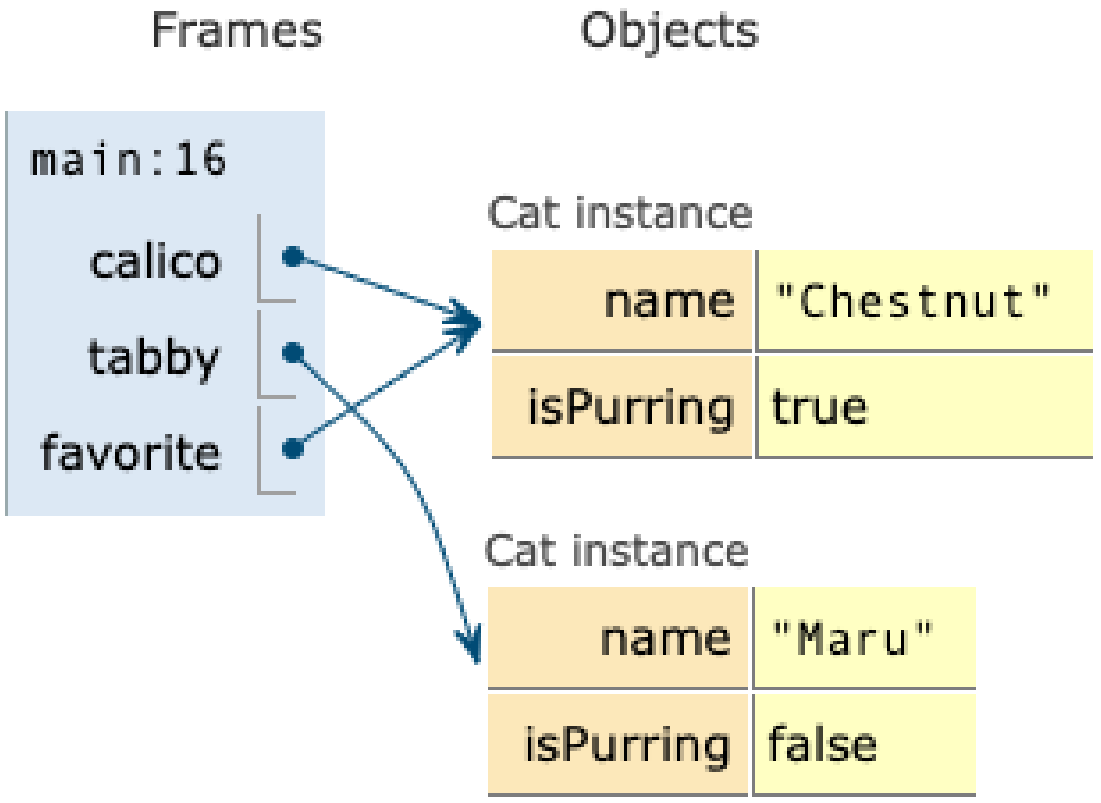
```
Cat calico = new Cat();
Cat tabby = new Cat();
Cat favorite = calico;
```

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



Updating object state

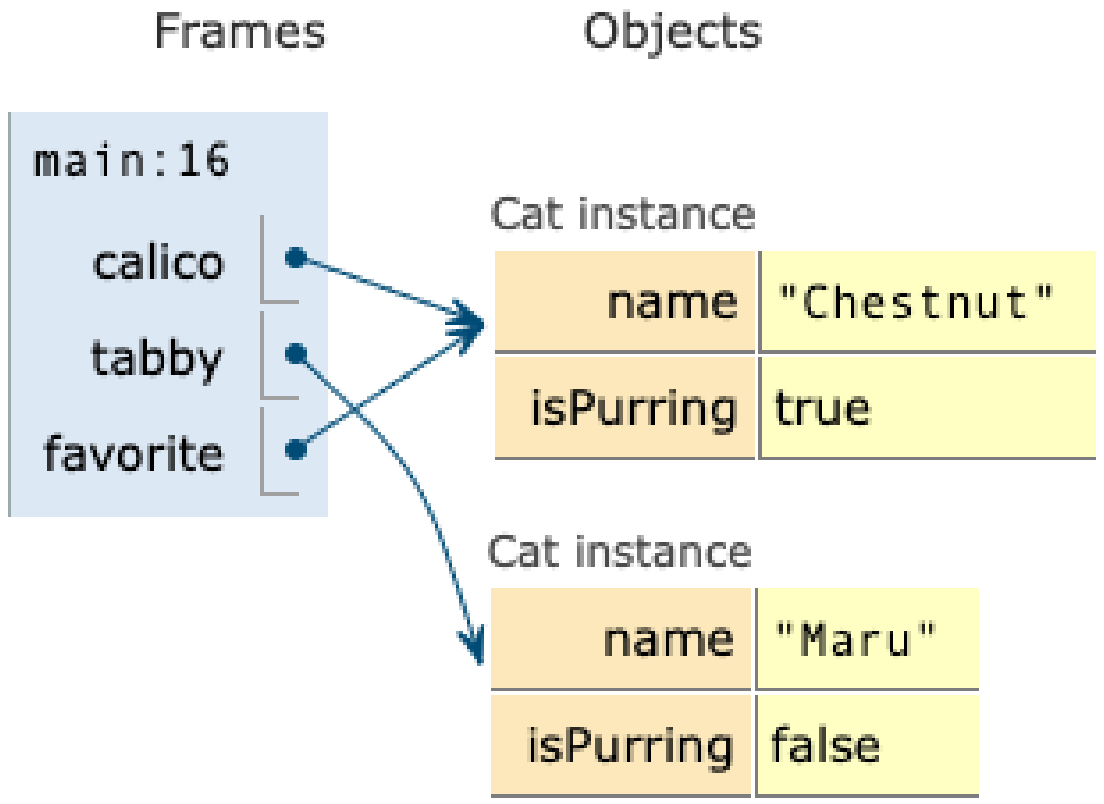
```
Cat calico = new Cat();
Cat tabby = new Cat();
Cat favorite = calico;

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



What get's printed?

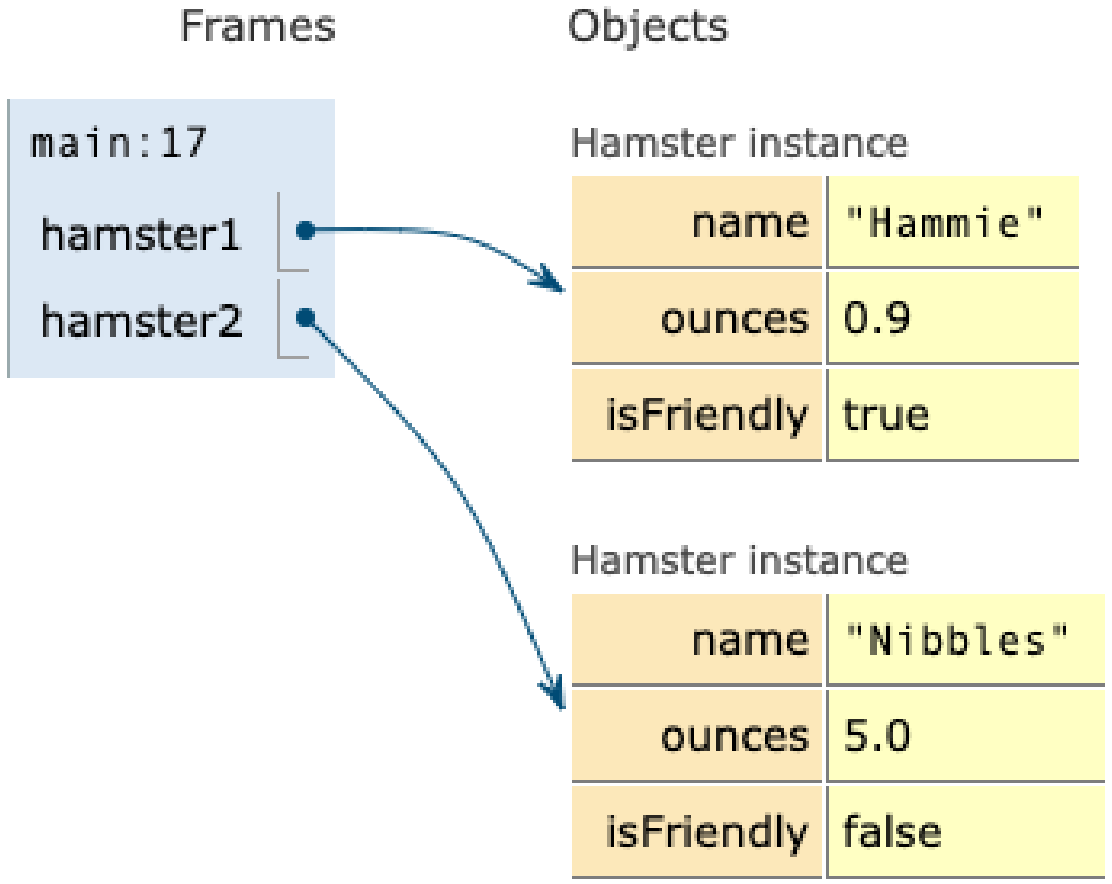
```
System.out.println(calico.name + "," + calico.isPurring);
System.out.println(tabby.name + "," + tabby.isPurring);
System.out.println(favorite.name + "," + favorite.isPurring);
```

```
Chestnut,true
Maru,false
Chestnut,true
```



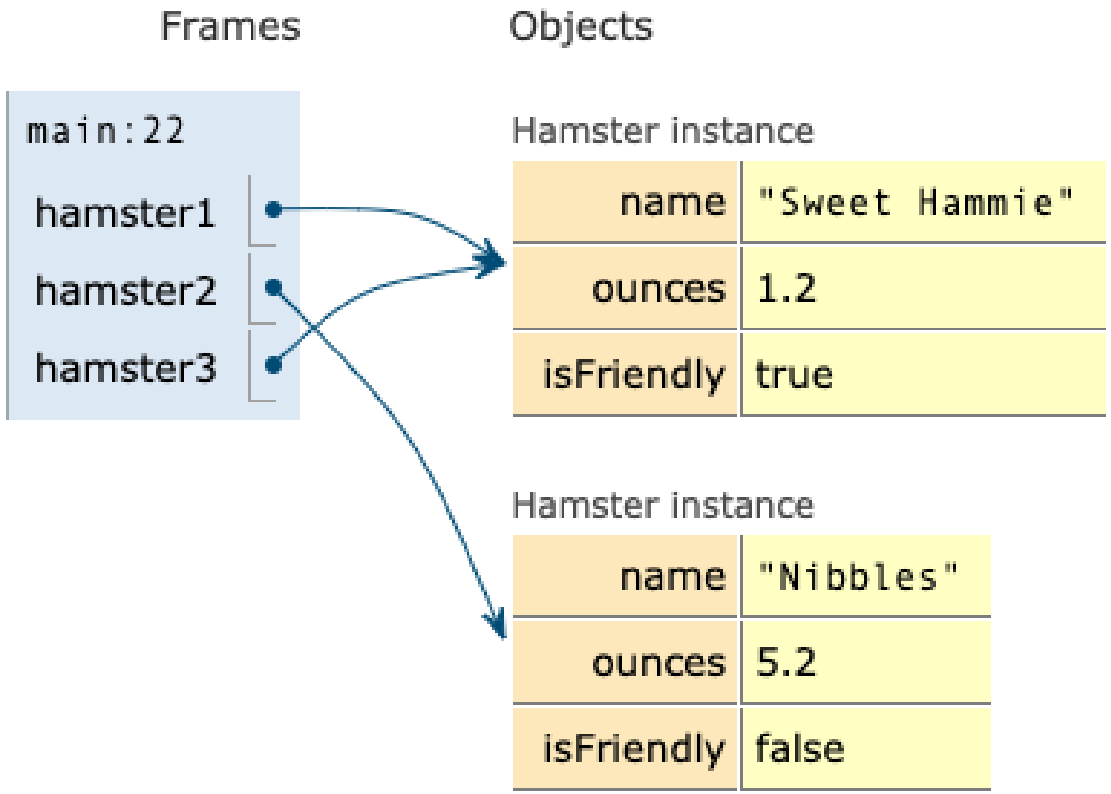
CHALLENGE

- Implement a class named `Hamster` with fields to store name, weight in ounces, and whether they are friendly.
- Implement a `main` method to instantiate two hamster and update their state as shown.



CHALLENGE:

- Edit the code to evolve the object state as shown in the diagram.
- Do not modify state using the variables `hamster1` or `hamster2`
- Use the variable `hamster3` to modify the `name` and `ounces`.



Solution

```
public class Hamster {

    String name;
    float ounces;
    boolean isFriendly;

    public static void main(String[] args) {
        Hamster hamster1 = new Hamster();
        Hamster hamster2 = new Hamster();

        hamster1.name = "Hammie";
        hamster1.ounces = 0.9f;
        hamster1.isFriendly = true;

        hamster2.name = "Nibbles";
        hamster2.ounces = Ωå5.2f;

        Hamster hamster3 = hamster1;
        hamster3.name = "Sweet Hammie";
        hamster3.ounces = 1.2f;

    }
}
```

Methods - Implementing Object Behavior