

# Some background before I start the lesson...

- Initial lessons use existing classes (String, ArrayList, Random, Swing/Graphics)
  - practice reading APIs
  - practice instantiating objects
  - practice invoking static and instance methods
- Subsequent lesson on defining a new Java class
  - Delay introduction of constructors and methods
  - Initial emphasize on object state and object references
  - Use visual debuggers to clarify object concepts, avoid common misconceptions



# Today's Lesson - Defining a new Java class

We've seen how to use existing Java core and utility classes (String, ArrayList, etc.) to solve some interesting problems.

Today we'll see how to define a **new** class to model some real world objects.



# Review: What is an object?

Objects have state (properties/data) and behavior (operation that access/modify state)

Object	State	Behavior
Mobile Phone	brand model is on volume	toggle on/off adjust volume send text
Zoom meeting	date time	schedule cancel



# Review: Java Data Types

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

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





# Review: Storing Random Coin Flips in an ArrayList

```
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());
}
```



# Defining a Java Class

- Template/blueprint for describing similar software objects.
- Define state (fields) and behavior (methods).

```
public class ClassName {  
  
    //Field declarations  
  
    //Method declarations  
  
}
```

# A class to model pet fish

## Objects

### Fish instance

age	15
isAggressive	false
species	"Goldfish"

### Fish instance

age	8
isAggressive	true
species	"Red Tail Shark"

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

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

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

## Java Expression    Heap (dynamic memory)

`new Fish()`

Fish instance

age	0
isAggressive	false
species	null

- Memory is allocated to store a value for each field

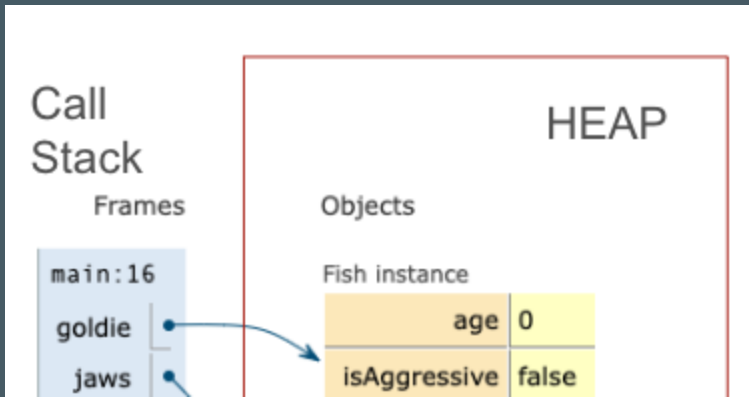
Fields are initialized with default values based on data type: int 0

# Reference Variable

## A reference variable:

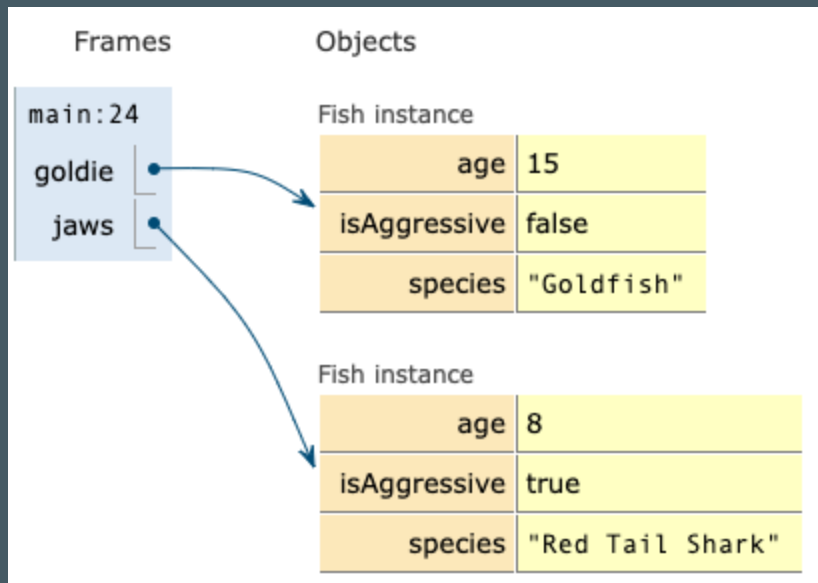
- Is declared with a reference data type (such as class **Fish**).
- Stores an object reference or `null`.

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



# Accessing an object's field

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



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

# NOTE : **String** is a reference data type

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

## String Literal (default view)

Fish instance

age	15
isAggressive	false
species	"Goldfish"

Fish instance

age	8
isAggressive	true
species	"Red Tail Shark"

## String Reference

Objects

Fish instance

age	15
isAggressive	false
species	

String  
"Goldfish"

Fish instance

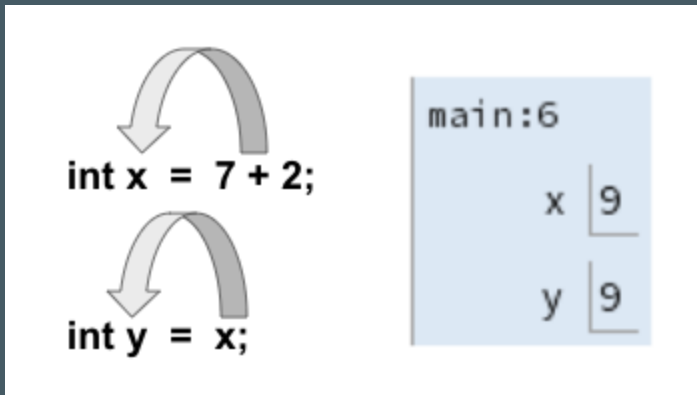
age	8
isAggressive	true
species	

String  
"Red Tail Shark"



# 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.



# CHALLENGE

Consider the following code:

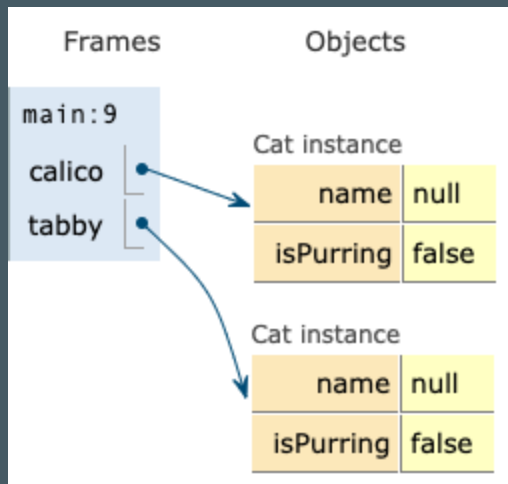
```
public class Cat {  
  
    String name;  
    boolean isPurring;  
  
    public static void main(String[] args) {  
        Cat calico = new Cat();  
        Cat tabby = new Cat();  
        Cat favorite = calico;  
  
        tabby.name = "Maru";  
        calico.name= "Chestnut";  
        favorite.isPurring = true;  
  
        System.out.printf("calico: %s %b%n", calico.name, calico.isPurring);  
    }  
}
```

# `new Cat()` creates an instance

<details> <summary>

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

</summary>



</details>

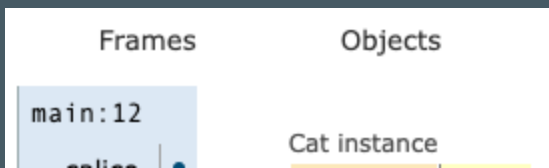
# Multiple variables can reference the same object

<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>



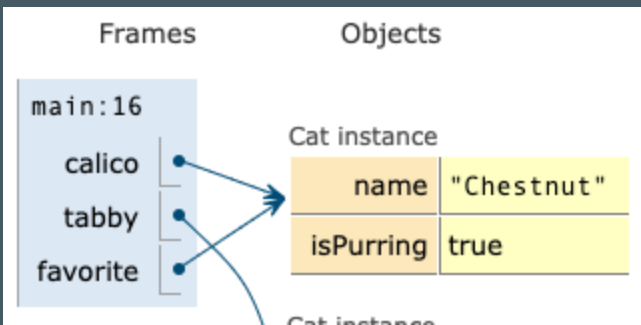
# Updating object state

<details> <summary>

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

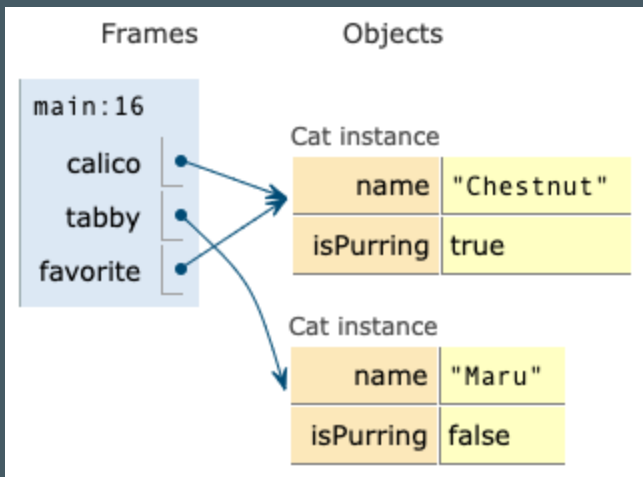
```
tabby.name = "Maru";  
calico.name= "Chestnut";  
favorite.isPurring = true;
```

</summary>



# 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 unnecessary field assignments (consider default initialization).
- Step through with the debugger to confirm your code is correct.

