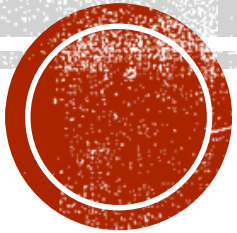
OBJECTS, CONSTRUCTORS

OOP Lecture 6



OBJECTS AND REFERENCES: OUTLINE

- Variables of a Class Type
- Defining an equals Method for a Class
- Boolean-Valued Methods
- Parameters of a Class Type

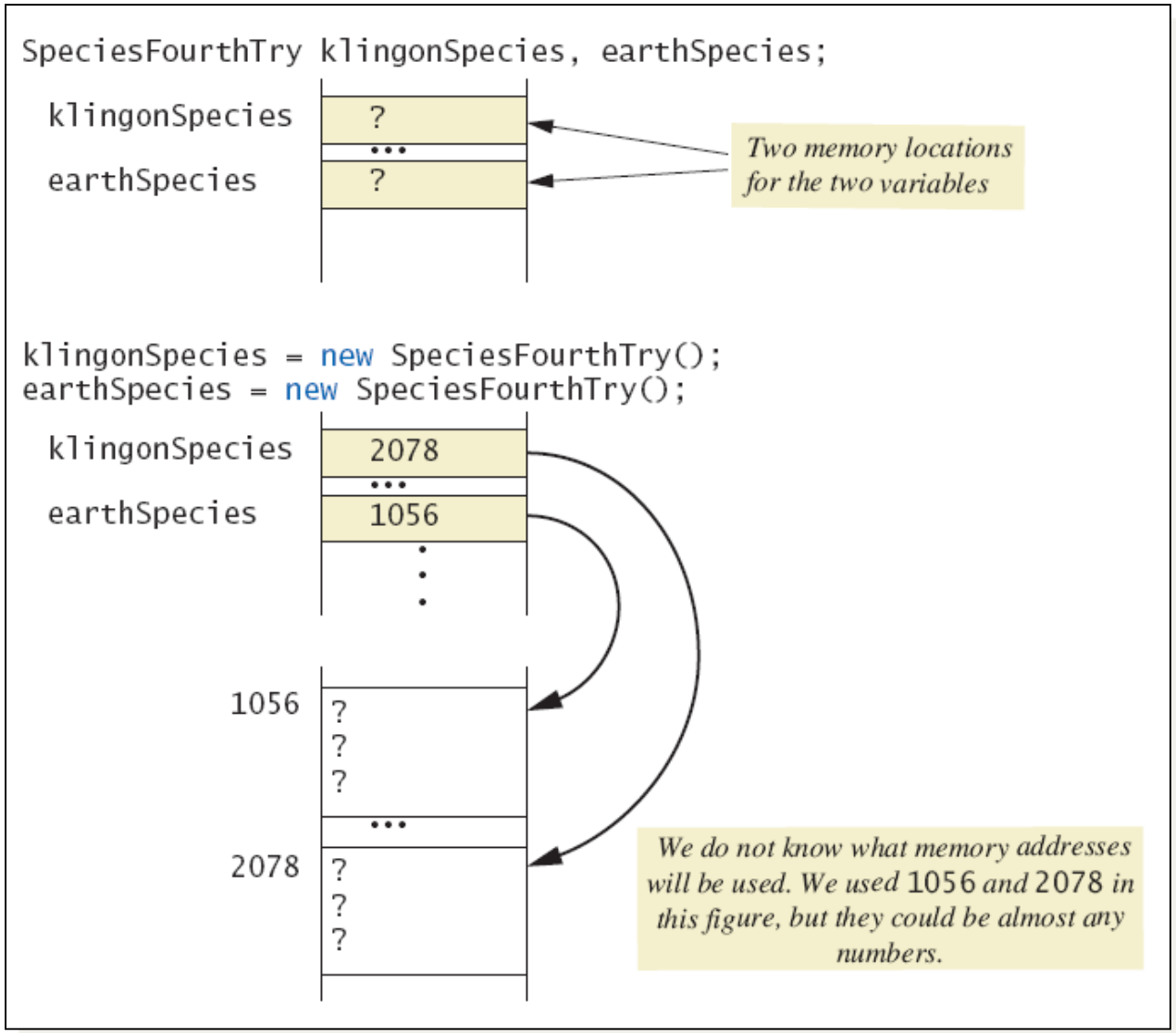
VARIABLES OF A CLASS TYPE

- All variables are implemented as a memory location
- Data of *primitive type* stored in the memory location assigned to the variable
- Variable of *class type* contains memory address of object named by the variable

VARIABLES OF A CLASS TYPE

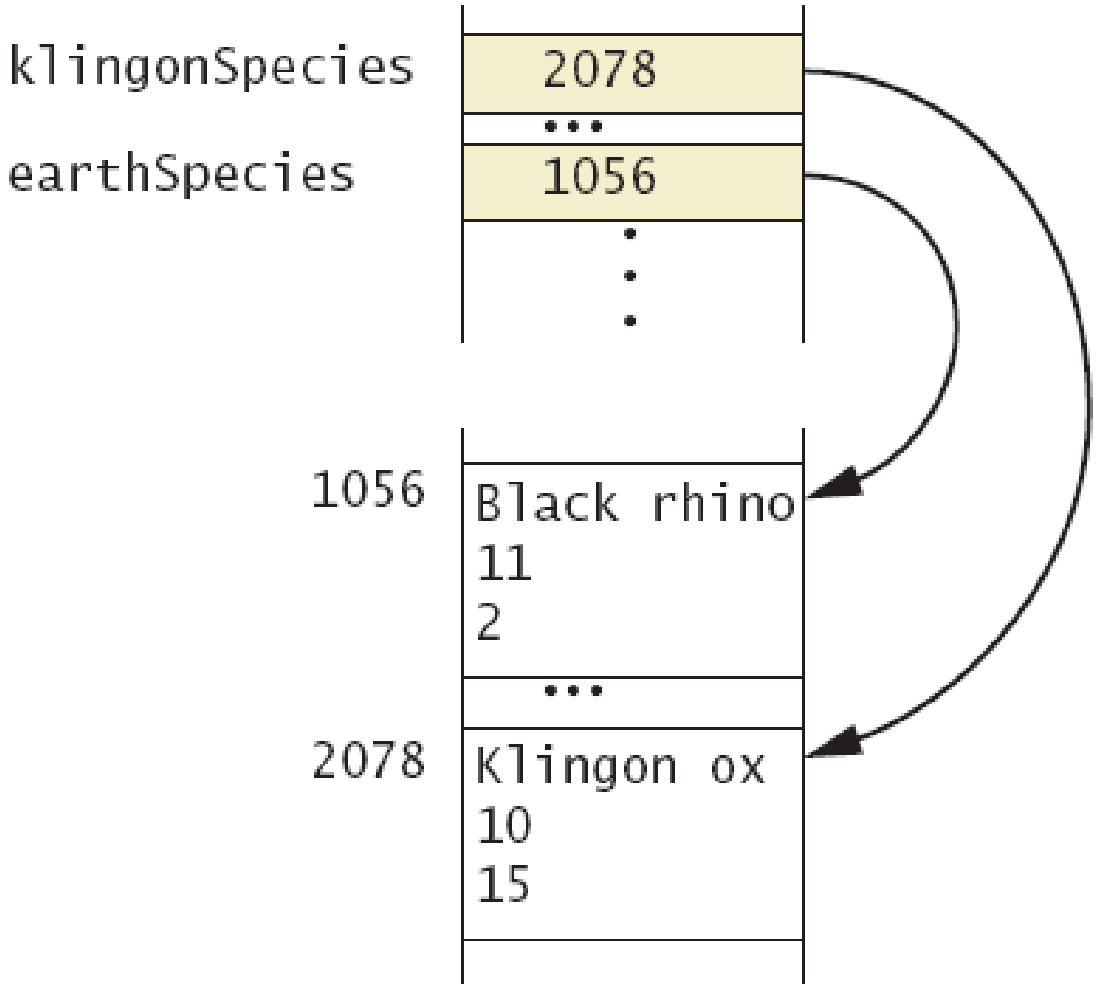
- Object itself not stored in the variable
  - Stored elsewhere in memory
  - Variable contains address of where it is stored
- Address of this memory location is called a *reference* to the object
- A *reference type* variable holds references (memory addresses) of objects
  - This makes memory management of class types more efficient

VARIABLES OF A CLASS TYPE



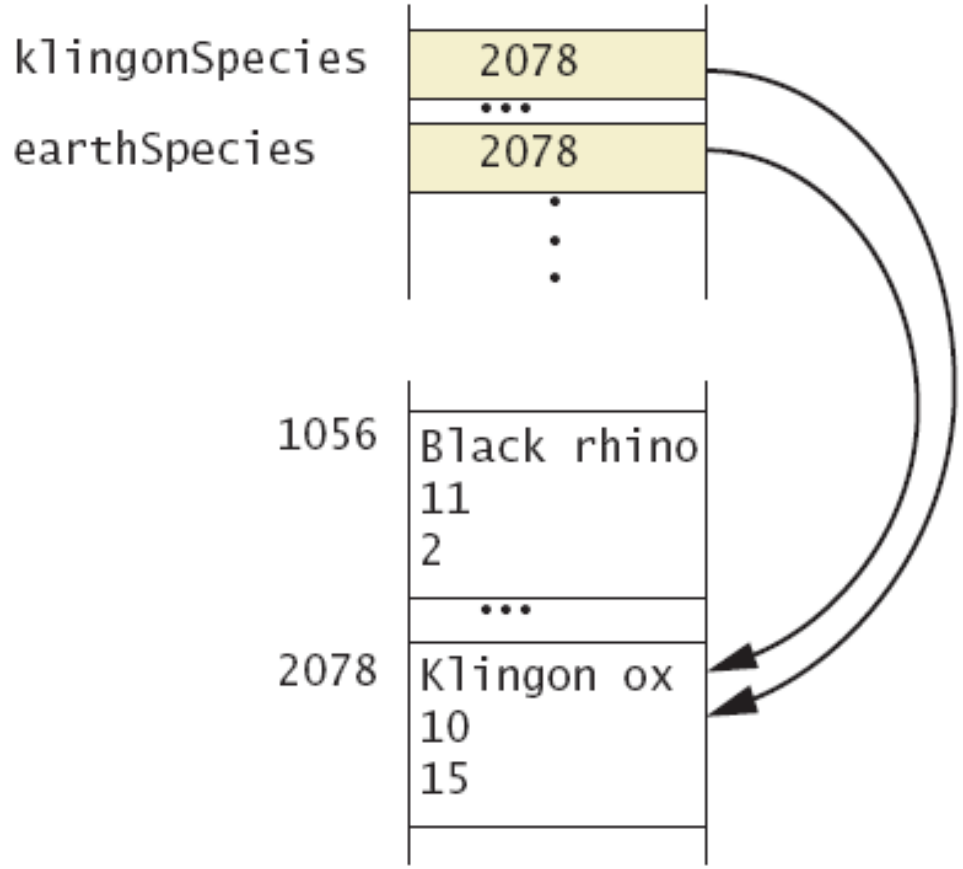
VARIABLES OF A CLASS TYPE

```
klingonSpecies.setSpecies("Klingon ox", 10, 15);
earthSpecies.setSpecies("Black rhino", 11, 2);
```



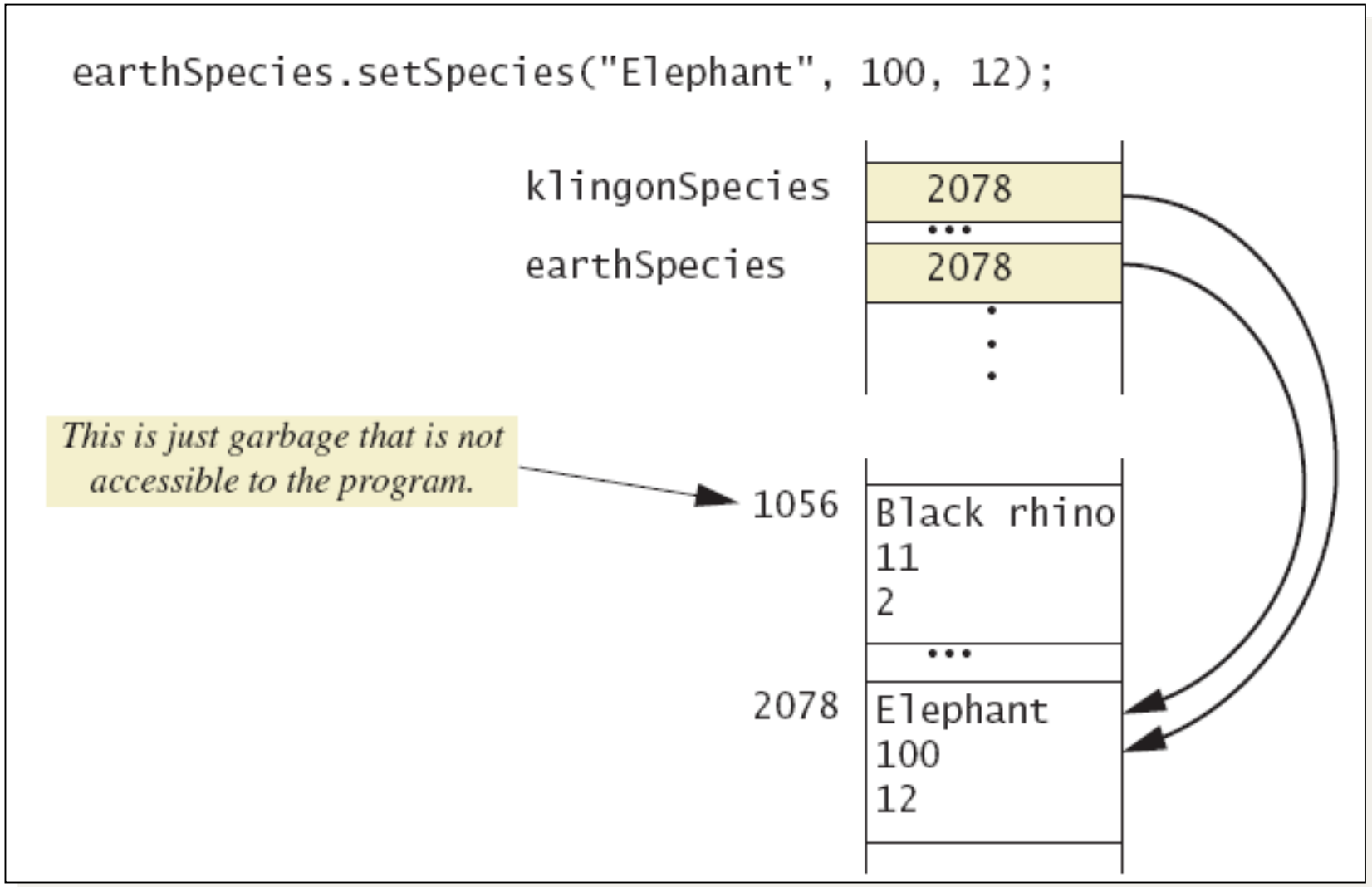
VARIABLES OF A CLASS TYPE

```
earthSpecies = klingonSpecies;
```



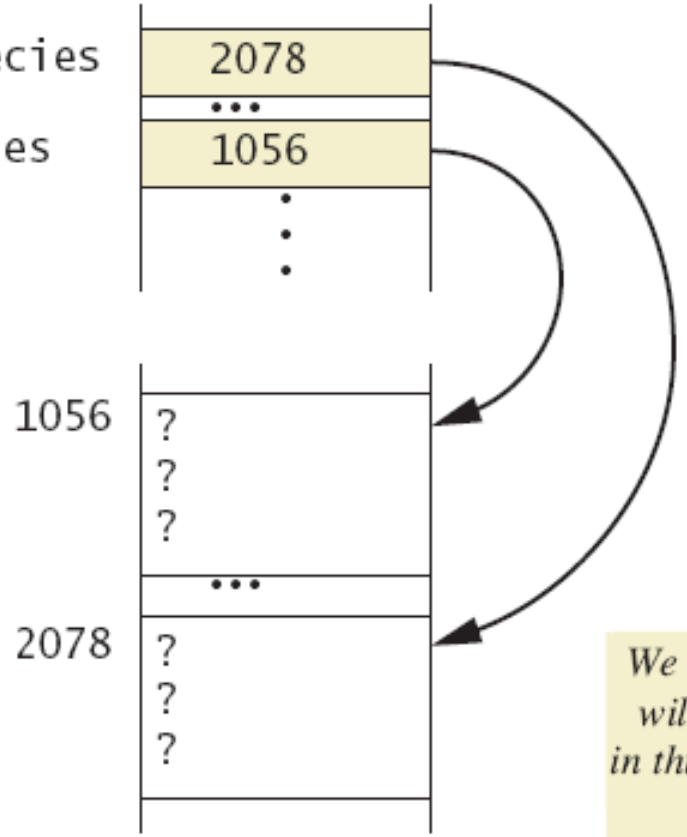
`klingonSpecies` *and* `earthSpecies` *are now two names for the same object.*

VARIABLES OF A CLASS TYPE



USE OF == WITH VARIABLES OF A CLASS TYPE

```
klingonSpecies = new SpeciesFourthTry();
earthSpecies = new SpeciesFourthTry();
```

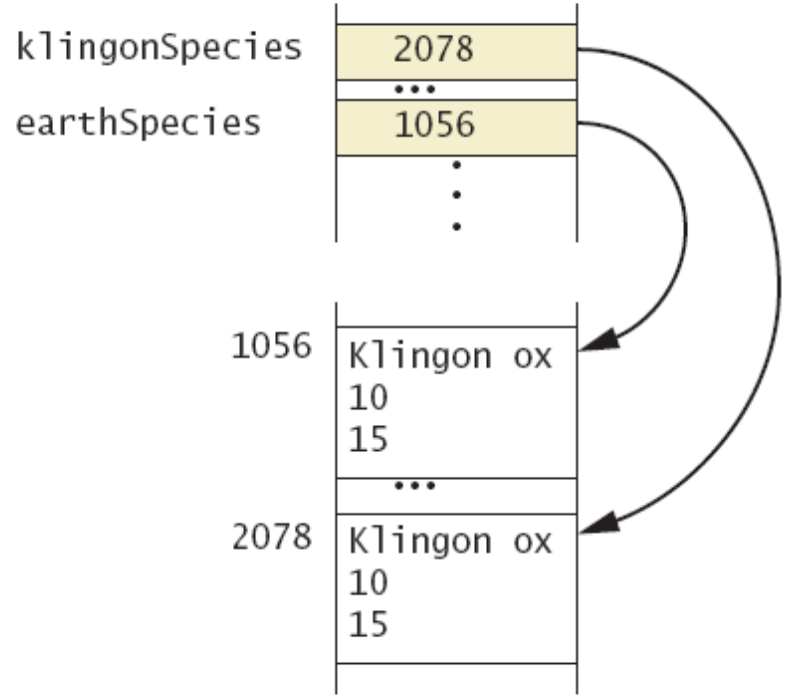


We do not know what memory addresses will be used. We used 1056 *and* 2078 *in this figure, but they could be almost any numbers.*

VARIABLES OF A CLASS TYPE

Dangers of using == with objects

```
klingonSpecies.setSpecies("Klingon ox", 10, 15);
earthSpecies.setSpecies("Klingon ox", 10, 15);
```




```
if (klingonSpecies == earthSpecies)
    System.out.println("They are EQUAL.");
else
    System.out.println("They are NOT equal.");
```

The output is `They are Not equal`, *because* 2078 *is not equal to* 1056.

== here checks only to see whether the memory addresses are equal.

DEFINING AN `EQUALS` METHOD

- As demonstrated by previous figures
  - We cannot use == to compare two objects
  - We must write a method for a given class which will make the comparison as needed
- The `equals` for this class method used same way as `equals` method for `String`

BOOLEAN-VALUED METHODS

- Methods can return a value of type **boolean**
- Use a **boolean** value in the **return** statement

```
/**
 Precondition: This object and the argument otherSpecies
 both have values for their population.
 Returns true if the population of this object is greater
 than the population of otherSpecies; otherwise, returns false.
*/
public boolean isPopulationLargerThan(Species otherSpecies)
{
    return population > otherSpecies.population;
}
```

UNIT TESTING

- A methodology to test correctness of individual units of code
  - Typically methods, classes
- Collection of unit tests is the **test suite**
- The process of running tests repeatedly after changes are make sure everything still works is **regression testing**

CONSTRUCTORS: OUTLINE

- Defining Constructors
- Calling Methods from Constructors
- Calling a Constructor from Other Constructors

DEFINING CONSTRUCTORS

- A special method called when instance of an object created with new
  - Create objects
  - Initialize values of instance variables
- Can have parameters
  - To specify initial values if desired
- May have multiple definitions
  - Each with different numbers or types of parameters

DEFINING CONSTRUCTORS

- Example class to represent pets
- Class Diagram for Class `Pet`
- Class diagram does not include constructors

```
Pet

- name: String
- age: int
- weight: double

+ writeOutput(): void
+ setPet(String newName, int newAge, double newWeight): void
+ setName(String newName): void
+ setAge(int newAge): void
+ setWeight(double newWeight): void
+ getName(): String
+ getAge(): int
+ getWeight(): double
```

DEFINING CONSTRUCTORS

- Note different constructors
  - Default
  - With 3 parameters
  - With String parameter
  - With double parameter
- Constructor without parameters is the default constructor
  - Java will define this automatically if the class designer does not define any constructors
  - If you <u>do</u> define a constructor, Java will <u>not</u> automatically define a default constructor
- Usually default constructors not included in class diagram

EXAMPLE: CONSTRUCTOR

```
public class Pet{
   private String name;
   private int age;     //in years
   private double weight;//in pounds

   public Pet(String initialName, int initialAge,  double initialWeight)   {
      name = initialName;
      if ((initialAge < 0) || (initialWeight < 0))   {
         System.out.println("Error: Negative age or weight.");
         System.exit(0);
      }
      else  {
         age = initialAge;
         weight = initialWeight;
      }
   }
```

```
public Pet( )
  {
     name = "No name yet.";
     age = 0;
     weight = 0;
  }
```

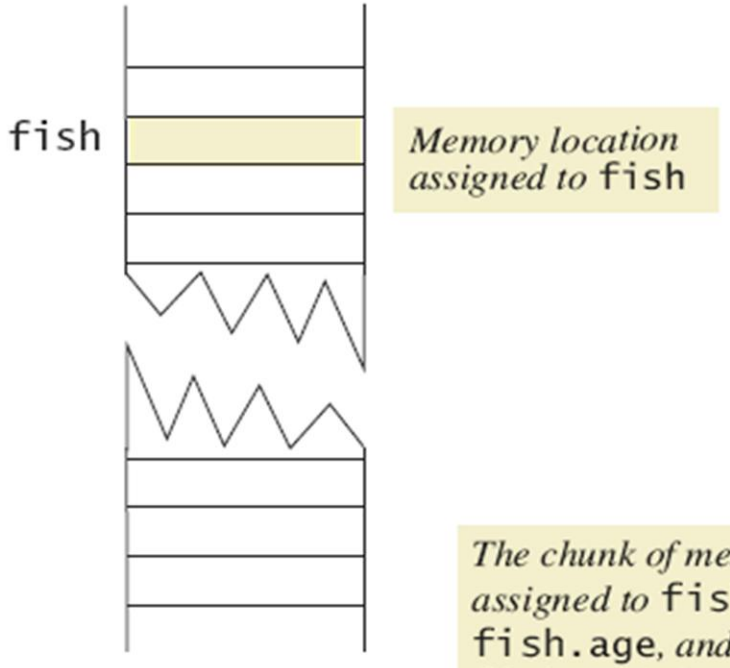
DEFINING CONSTRUCTORS

```
Pet fish;
```

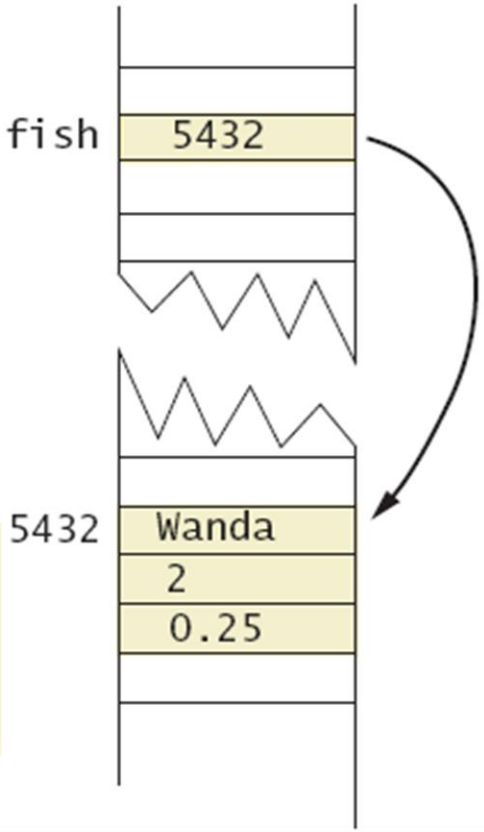
Assigns a memory location to `fish`

```
fish = new Pet();
```

Assigns a chunk of memory for an object of the class `Pet`*—that is, memory for a name, an age, and a weight—and places the address of this memory chunk in the memory location assigned to* `fish`



Memory location assigned to `fish`



The chunk of memory assigned to `fish.name`, `fish.age`, *and* `fish.weight` *might have the address* 5432.

CALLING METHODS FROM OTHER CONSTRUCTORS

- Constructor can call other class methods

```
public Pet(String initialName, int initialAge,
           double initialWeight)
{
    setPet(initialName, initialAge, initialWeight);
}
```

CALLING CONSTRUCTOR FROM OTHER CONSTRUCTORS

- A constructor can call another constructor by using the keyword this.
- In the other constructors use the this reference to call initial constructor

```
Class Pet{
Private String name;
Private int age;
Private double weight;
……
Public Pet(int initialAge)
{
        this("No name yet", initialAge, 0);
}
```

```
public Pet(String initialName, int initialAge,
           double initialWeight)
{
    setPet(initialName, initialAge, initialWeight);
}
```

STATIC VARIABLES

- Static variables are shared by all objects of a class
  - Variables declared `static final` are considered constants – value cannot be changed
- Variables declared `static` (without `final`) can be changed
  - Only one instance of the variable exists
  - It can be accessed by all instances of the class

  Example:
  - **public static final double LENGTH = 5.0;**
  - **private static int numberOfstudents;**
  - Static variables that are not constants should be private.

STATIC VARIABLES

- Static variables also called *class variables*
  - Contrast with *instance variables*
- Do not confuse class variables (i.e. static variables) with variables of a class type
- Both static variables and instance variables are sometimes called *fields* or *data members*

STATIC METHODS

- Some methods may have no relation to any type of object
- Static method declared in a class
  - Can be invoked without using an object
  - Instead use the class name

```java
/**
 Class of static methods to perform dimension conversions.
*/
public class DimensionConverter
{
    public static final int INCHES_PER_FOOT = 12;

    public static double convertFeetToInches(double feet)
    {
        return feet * INCHES_PER_FOOT;
    }

    public static double convertInchesToFeet(double inches)
    {
        return inches / INCHES_PER_FOOT;
    }
}
```

A static constant; it could be private here.

```java
double feet = DimensionConverter.convertInchesToFeet(53.7);
double inches = DimensionConverter.convertFeetToInches(2.6);
```

THE MATH CLASS

- This class is automatically provided when we use the Java language.
- Provides many standard mathematical methods
  - Automatically provided, no import needed
  - All these methods are static.
- Example methods.

Name	Description	Argument Type	Return Type	Example	Value Returned
pow	Power	double	double	Math.pow(2.0,3.0)	8.0
abs	Absolute value	int, long, float, or double	Same as the type of the argument	Math.abs(-7) Math.abs(7) Math.abs(-3.5)	7 7 3.5
max	Maximum	int, long, float, or double	Same as the type of the arguments	Math.max(5, 6) Math.max(5.5, 5.3)	6 5.5

THE MATH CLASS

- Example methods,

Name	Description	Argument Type	Return Type	Example	Value Returned
`min`	Minimum	`int`, `long`, `float`, or `double`	Same as the type of the arguments	`Math.min(5, 6)` `Math.min(5.5, 5.3)`	5 5.3
`round`	Rounding	`float` or `double`	`int` or `long`, respectively	`Math.round(6.2)` `Math.round(6.8)`	6 7
`ceil`	Ceiling	`double`	`double`	`Math.ceil(3.2)` `Math.ceil(3.9)`	4.0 4.0
`floor`	Floor	`double`	`double`	`Math.floor(3.2)` `Math.floor(3.9)`	3.0 3.0
`sqrt`	Square root	`double`	`double`	`sqrt(4.0)`	2.0

RANDOM NUMBERS

- `Math.random()` returns a random double that is greater than or equal to zero and less than 1
- Java also has a `Random` class to generate random numbers
- Can scale using addition and multiplication; the following simulates rolling a six sided die

```
int die = (int) (6.0 * Math.random()) + 1;
```

  - Multiplying a random number by 6.0 gives a value in the range >= 0 and <6.0.
  - Typecasting it to int gives 0,1,2,3,5,6.
  - Adding 1 results in a random integer in the range 1- 6.

WRAPPER CLASSES

- Recall that arguments of primitive type treated differently from those of a class type
  - May need to treat primitive value as an object
- Java provides *wrapper classes* for each primitive type
- Wrapper classes define methods that can act on values

WRAPPER CLASSES

- Allow programmer to have an object that corresponds to value of primitive type
- Contain useful predefined constants and methods
- Wrapper classes have no default constructor
  - Programmer must specify an initializing value when creating new object
  - Example: Integer n = new Integer(10); - **Boxing**
  - int i = n.intValue(); - **Unboxing**
- Wrapper classes have no `set` methods

WRAPPER CLASSES

- Static methods in class `Character`

Name	Description	Argument Type	Return Type	Examples	Return Value
`toUpperCase`	Convert to uppercase	`char`	`char`	`Character.toUpperCase('a')` `Character.toUpperCase('A')`	`'A'` `'A'`
`toLowerCase`	Convert to lowercase	`char`	`char`	`Character.toLowerCase('a')` `Character.toLowerCase('A')`	`'a'` `'a'`
`isUpperCase`	Test for uppercase	`char`	`boolean`	`Character.isUpperCase('A')` `Character.isUpperCase('a')`	`true` `false`

WRAPPER CLASSES

- Static methods in class `Character`

Name	Description	Argument Type	Return Type	Examples	Return Value
`isLowerCase`	Test for lowercase	`char`	`boolean`	`Character.isLowerCase('A')` `Character.isLowerCase('a')`	`false` `true`
`isLetter`	Test for a letter	`char`	`boolean`	`Character.isLetter('A')` `Character.isLetter('%')`	`true` `false`
`isDigit`	Test for a digit	`char`	`boolean`	`Character.isDigit('5')` `Character.isDigit('A')`	`true` `false`
`isWhitespace`	Test for whitespace	`char`	`boolean`	`Character.isWhitespace(' ')` `Character.isWhitespace('A')`	`true` `false`

Whitespace characters are those that print as white space, such as the blank, the tab character (`'\t'`), and the line-break character (`'\n'`).

OVERLOADING: OUTLINE

- Overloading Basics
- Overloading and Automatic Type Conversion
- Overloading and the Return Type
- Programming Example: A Class for Money

OVERLOADING BASICS

- When two or more methods have same name within the same class
- Java distinguishes the methods by number and types of parameters
  - If it cannot match a call with a definition, it attempts to do type conversions
- A method's name and number and type of parameters is called the *signature*

OVERLOADING AND TYPE CONVERSION

- Overloading and automatic type conversion can conflict
- Remember the compiler attempts to overload before it does type conversion
- Use descriptive method names, avoid overloading

```
public Pet(String initialName, int initialAge,
           double initialWeight)

public Pet(String Parm_1, int Parm_2, int
           Parm_3)

public Pet(String newName, int newAge,
           double newWeight)
```

```
Pet myDog = new Pet("Cha Cha", 2, 3);
```

OVERLOADING AND RETURN TYPE

- You must not overload a method where the only difference is the type of value returned

```
/**
 Returns the weight of the pet.
*/
public double getWeight()

/**
 Returns '+' if overweight, '-' if
 underweight, and '*' if weight is OK.
*/
public char getWeight()
```