Objects

Overview

- Most real or imaginary world entities have properties and behavior
- In this module, we will learn how to represent the properties (or attributes) and the behavior of the entities that our program will manipulate
- Example:
  - Entity: `student`
  - Properties: `name, age, height, etc.`
  - Behavior: `play, read, write, speak, etc.`

Learning Objectives

- To be able to create and initialize objects
- To be able to call methods without parameters
- To be able to call methods with parameters
- To be able to call methods that return a value
- To be able to manipulate `String` values

Modeling with objects

- Objects are used to model real-worlds entities
- An **object** has some **property/ies** or **attribute/s** and **behavior/s**
    - An attribute describes the object
    - A behavior tells us what the object does: **methods**

Figure 2: Pictures of cats (cat objects)

Objects in Java

- Objects are created from a class definition
- A **class is a template for** creating objects
- Objects are instances of a class
- Each class has **constructors** that are used to **initialize** the **attributes** in a newly created object
- The constructor and the class have the same name

Class	Object
Cat	Garfield the cat

Objects in Java

- To create an object you write

constructor

```
ClassName variableName = new ClassName(arguments);
```

Example:

A cat has the following attributes: `name`, `color`

To create a new `orange Cat` named “`Garfield the cat`” you write

```
Cat garfield = new Cat("Garfield the cat", "orange");
```

- class name
- variable name
- constructor
- Arguments: initial values (name, color)

Creating objects

- We can create more than one objects of the same class

```
Cat garfield = new Cat("Garfield the cat", "orange");
Cat myCat = new Cat("mona", "yellow");
Cat yourCat = new Cat("midnight", "black");
```

- Our program will manipulate the following objects (cats)

Object name	name	Color
garfield	Garfield the cat	orange
myCat	mona	yellow
yourCat	midnight	black

Constructors

- A class can have more than one constructor
- Defining more than one constructors is called **overloading** the constructor
- The **no-argument constructor i**s the constructor without parameters
- The **no-argument** constructor usually sets the attributes of the object to default values

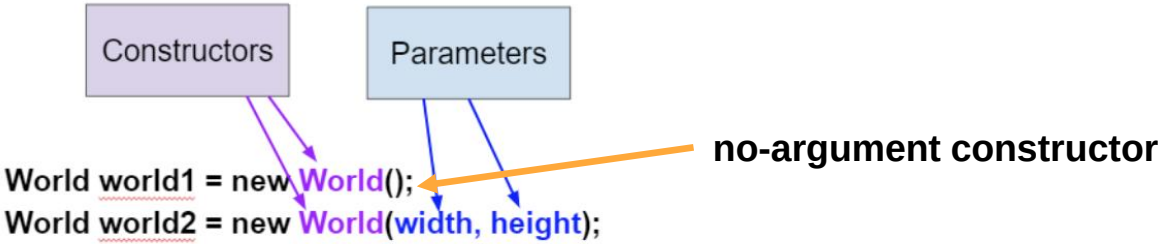

Figure 1: Two overloaded World constructors

Constructors signature

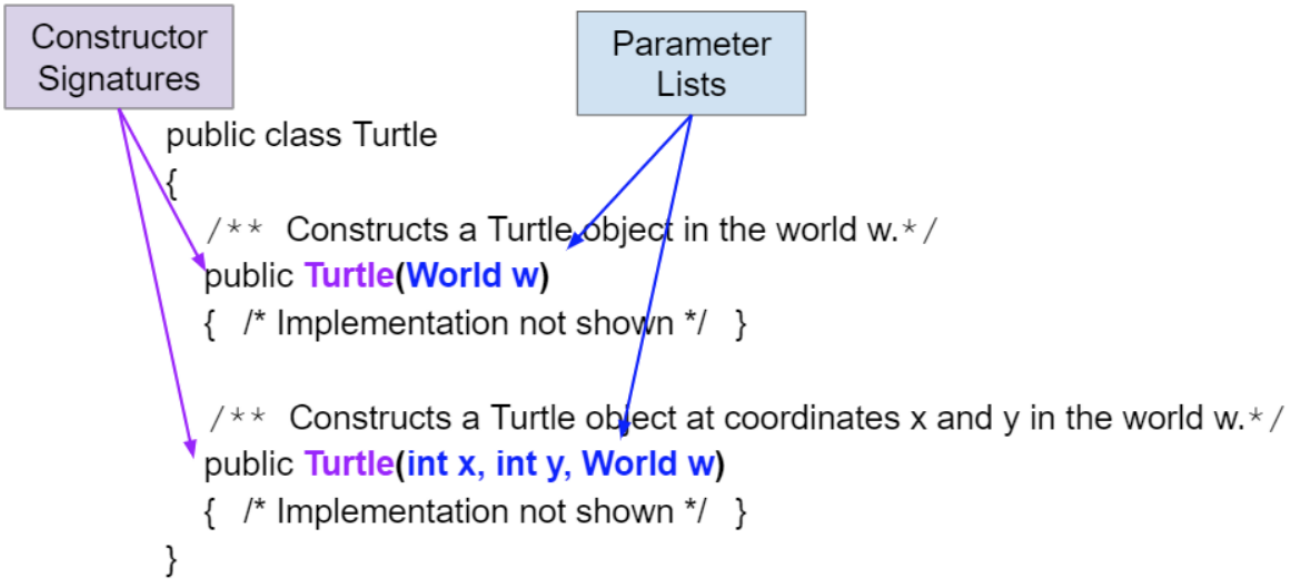

Figure 3: Turtle Class Constructor Signatures and Parameters

Formal vs actual parameters

- When calling the constructor to create a new object, you must pass **actual parameters**
- Formal parameters are included I the constructor signature

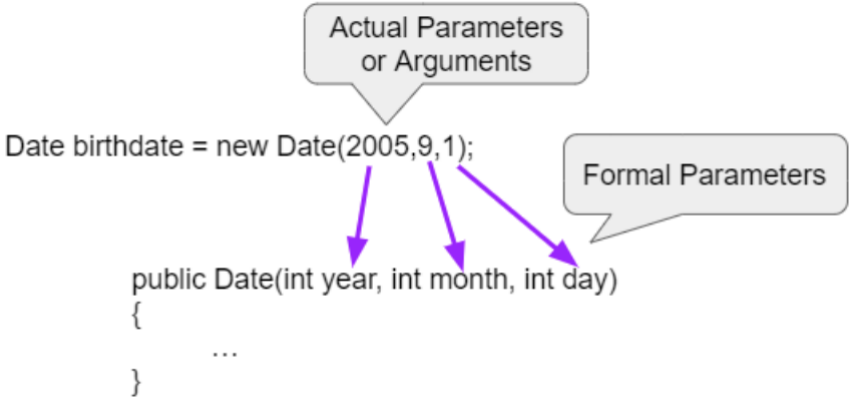

Figure 5: Parameter Mapping

Objects behavior

- Behavior of objects is defined inside methods
- **Methods** are a set of instructions that define behaviors for all objects of a class
- An object method ***must*** be called on an object of the class that the method is defined in
- Object methods work (modify) with the **attributes** of the object
- Object methods are also called **non-static methods**

Calling Methods

- To call a method on an object, you write

```
Object_Name.method_name(parameters);
```

Example:

```
Cat garfield = new Cat();
garfield.play();
garfield.move(destination);
```

Method without parameters

Method with parameters

Calling methods that return values

- A method has a return type
- A **void method** does not return a value
- **Get methods** return the value of instance variables
- When using a get method
    - you should save what it returns in a variable or
    - You should use the value in some way for example by printing it out

```
Turtle yertle = new Turtle(world);
int width = yertle.getWidth();
int height = yertle.getHeight();
System.out.println("Yertle's width is: " + width);
System.out.println("Yertle's height is: " + height);
System.out.println("Yertle's x position is: " + yertle.getXPos() );
System.out.println("Yertle's y position is: " + yertle.getYPos() );
```

Strings

- **Strings** are objects of the String class
- Strings hold sequences of characters (a, b, c, $, etc)
- Write `String variable_name;` to declare a string object
- A string like other objects can be initialized to a `null reference`
- `A null reference` means that the variable does not refer to a space in memory
    - `String variable_name = null;` creates a null string object

String initialization

- There are two ways to initialize a string
- `String variable_name = new String(string_literal);`
    - Example: String name = new String("Lisa");

- `String variable_name = string_literal;`
    - Example: String name = "Lisa";

String operations

- **Concatenation**
- Use the **“+”** or **“+=”** operators to concatenate (combine) two Strings

```
String a = "Serena";
String b = " Williams";
String c = a + b;
System.out.println(c); // prints Serena Williams
```

String operations

- Using **“+”** or **“+=”** operators to append a primitive type value to a String will automatically convert that value to String

```
String a = "Serena";
String b = " Williams";
String c = a + b + 100;
System.out.println(c); // prints Serena Williams100
```

Aside: Object methods and '.'

- The + and += operator on strings is somewhat unique. Normally performing an operation on an object requires different syntax.

- Example: If we have

```
String a = "Serena";
String b = " Williams";
```

- We can do:

```
String c = a + b;  // assigns "Serena Williams"
```

- Or equivalently:

```
String c = a.concat(b); // assigns "Serena Williams"
```

There is NO space around the '.'

String methods

- `int length()` method returns the number of characters in the string, including spaces and special characters like punctuation

```
String a = "Serena";
a.length(); // returns 6
```

String methods

- `String substring(int from, int to)`
  - returns a new string with the characters in the current string starting with the character at the `from` index and ending at the character *before* the `to` index (if the `to` index is not specified it will contain the rest of the string)

```
String a = "Serena";
           0 1 2 3 4 5

String b = a.substring(0, 3);      0 1 2
System.out.println(b); // prints "Ser"
String c = a.substring(3);             3 4 5
System.out.println(c); // prints "ena"
```

String methods

- **`int indexOf(String str)`** method searches for the string str in the current string and returns the index of the beginning of str in the current string or -1 if it isn't found

```
String a = "Serena";
           012345

int x = a.indexOf("er");  // returns 1
int y = a.indexOf("ena"); // returns 3
int z = a.indexOf("sa");  // returns -1
```

Comparing Strings

- Strings (and objects) **cannot** be compared using operators like `==` and `<` or `>`
- The method `compareTo` compares two strings character by character.
    - If they are **equal**, it returns **0**
    - If the **first string** is alphabetically ordered **before** the **second string** it returns a **negative number**
    - If the **first string** is alphabetically ordered **after** the **second string**, it returns a **positive number**

Comparing Strings

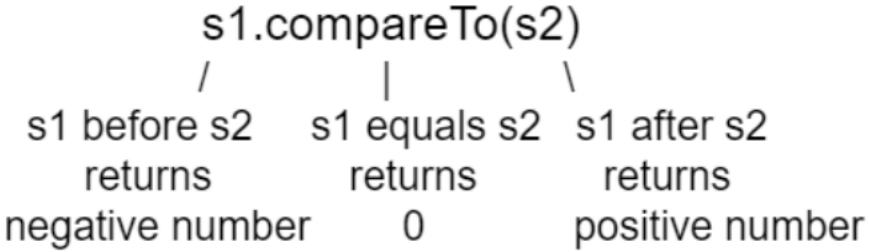

Figure 2: compareTo returns a negative or positive value or 0 based on alphabetical order

```
String a = "Serena";

String b = "Williams";
```

S comes before **W** in the alphabet

```
a.compareTo(b); // return -4 negative number
b.compareTo(a); // return 4 positive number
```

String equality

DO NOT USE ==

- The `equals` method compares the two strings character by character and returns true or false

```
String a = "Serena";
String b = "Williams";
a.equals(b); // returns false
a.equals(a); // returns true
```

- `compareTo`, `equals` and most string methods are case-sensitive.

```
"HI".equals("hi");  // returns false
```