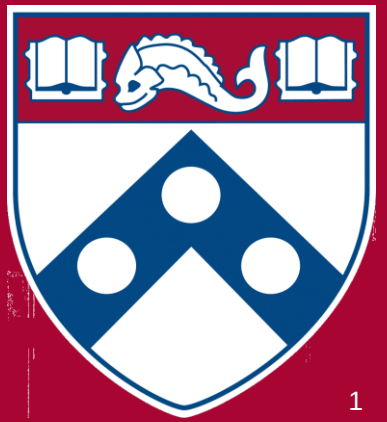
Classes & Methods



Overview

- Before we can manipulate real or imaginary world entities, we need to create them
- In this module, we will learn how to create and manipulate real or imaginary world entities

- Example:
    - Create an entity student with attributes name, and age, and with behavior dance, and play

Learning Objectives

- **To be able to write and use a class**
- To be able to write a class constructor
- To be able to write comments
- To be able to understand and write accessor and mutator methods
- To be able to write methods
- To be able to use static variables and methods
- To be able to understand variable scope
- To be able to understand and use the this keyword

Introduction

- A **class is a template for** creating objects
- A class defines a new **data type**
- A class defines the object's **attributes / properties** and **behavior**
- Object's **attributes** are implemented as **instance variables**
- Object's behavior are implemented as **methods**
- Objects are instances of a class

Class Design

- **Abstraction**: set of information properties relevant to a stakeholder about an entity
- Information Property (or property): a named, objective and quantifiable aspect of an entity
- Stakeholder: a real or imagined person (or a class of people) who is seen as the audience for, or user of the abstraction being defined

Class Design

- Entity: Movie
- Properties:
    - Title
    - Year
    - Length
    - Genre
    - Format
    - Price

Class Design

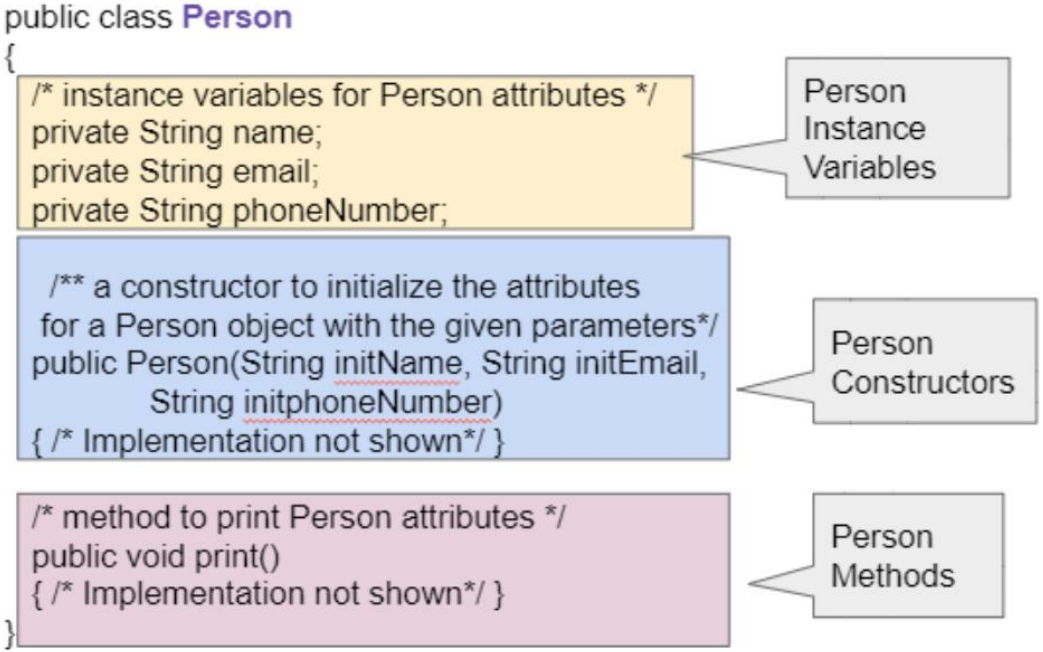
Movie			On-Line Customer		
Title (string)	Year (int)	Length (int)	Genre (string)	Format (string)	Price (double)
"Moneyball"	2011	133	"Sports"	"Blueray"	15.00
"Gone With the Wind"	1939	219	"Drama"	"DVD"	10.95
"Jurassic Park"	1993	127	"SciFi"	"DVD"	12.50
"Pirates of the Caribbean"	2003	143	"Comedy"	"Blueray"	17.50
"Sicko"	2007	116	"Documentary"	"Streaming"	11.75

Representing the Movie Abstraction using a Table

Content of a Class

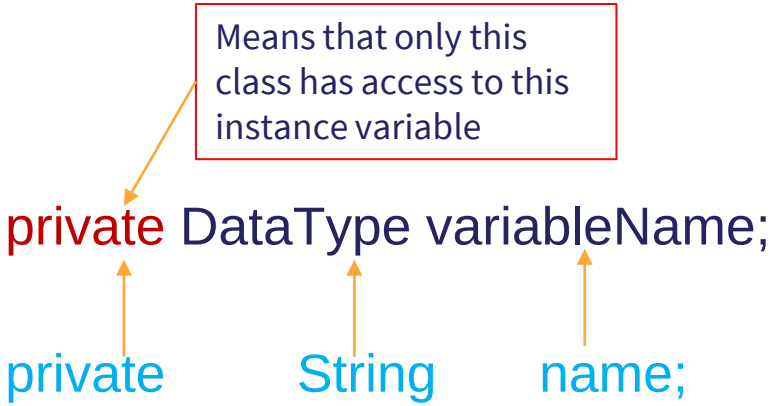
- A class contains
  - **Instance variables** (attributes) representing the properties of the abstraction
  - One or more **constructor(s)** to **initialize** the objects' **instance variables** (attributes)
  - Methods to implement the objects' behavior

Anatomy of a Class



Instance variables

- Listed at the top of the class definition
- To declare an instance variable, you write



Instance variables

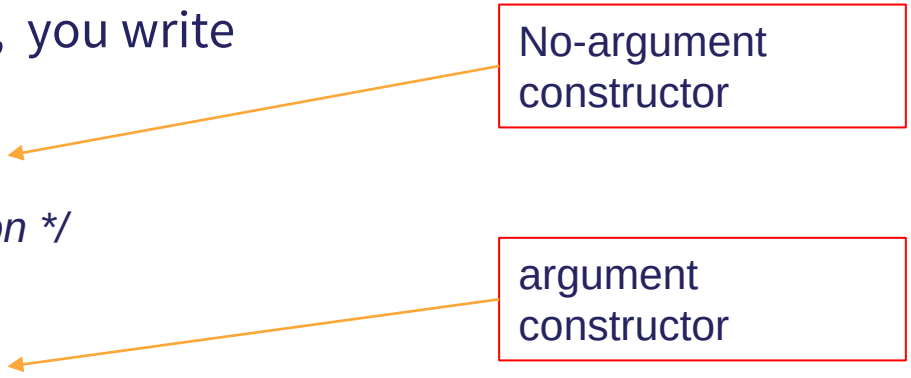
```
public class Movie {
    private String title;
    private int year;
    private int length;
    private String genre
    private String format
    private double price
            …

}
```

Learning Objectives

- To be able to write and use a class
- **To be able to write a class constructor**
- To be able to write comments
- To be able to understand and write accessor and mutator methods
- To be able to write methods
- To be able to use static variables and methods
- To be able to understand variable scope
- To be able to understand and use the this keyword

Constructors

- Set the **initial values** for the object's **instance variables**
- Constructors must have the same name as the class
- Constructors have no return type!
- To define a constructor, you write




```
public  className(){
   /* instance variable initialization */
}
Or
public  className(DataType1 parameter1, DataType2 parameter2, …){
    /* instance variable initialization */
}
```

No-argument constructor

- Default constructor (provided by Java)
- Initializes instance variables to default values
- Person class

```
public  Person(){
    name = "";
    email = "";
    phoneNumber = "";
}
```

default value: "" / empty string

Argument constructor

- Person class

```
public Person(String initName, String initEmail,
                              String initPhone) {
    name = initName;
    email = initEmail;
    phoneNumber = initPhone;
}
```

Initializes instance variables to parameters /arguments

Learning Objectives

- To be able to write and use a class
- To be able to write a class constructor
- **To be able to write comments**
- To be able to understand and write accessor and mutator methods
- To be able to write methods
- To be able to use static variables and methods
- To be able to understand variable scope
- To be able to understand and use the this keyword

Comments

- There are 3 types of comments in Java:
  - *// Single line comment*
  - */* Multiline comment **/*
  - */*** Documentation comment **/*

Preconditions and Postconditions

- Precondition:
  - A condition that must be true for your method to work
- Postcondition:
  - A condition that is true after running the method

```
/**
  * Constructor that takes the x and y position for the
  * turtle
  * Preconditions: parameters x and y are coordinates from 0 to
  *    the width and height of the world.
  * Postconditions: the turtle is placed in (x,y) coordinates
  * @param x the x position to place the turtle
  * @param y the y position to place the turtle
  */
 public Turtle(int x, int y)
 {
   xPos = x;
   yPos = y;
 }
```

Learning Objectives

- To be able to write and use a class
- To be able to write a class constructor
- To be able to write comments
- **To be able to understand and write accessor and mutator methods**
- To be able to write methods
- To be able to use static variables and methods
- To be able to understand variable scope
- To be able to understand and use the this keyword

Special methods

- **Accessor methods** to **retrieve** and **return** the **value** of the **instance variables**
- **Mutator methods** to **change** (update) the **value** of the **instance variables**
- **Main method** used to **test** your **class** (execute your code). There can be **only one main method** inside a class

Accessor Methods

- Getter methods
- Return the value of each instance variables
- To define a getter method, you write

```
public  VariableType getVariableName(){
   return variableName
}
```

```
public  String getName(){
   return name;
}
```

Person Class:
name (String) instance variable
getter

Mutator Methods

- Setter method
- Change the value of a (private) instance variable
- To define a setter method, you write

```
public  void setVariableName(VariableType value){
   variableName = value;
}
```

```
public  void setName(String newName){
   name = newname;
}
```

Person Class:
name (String) instance variable
setter

Learning Objectives

- To be able to write and use a class
- To be able to write a class constructor
- To be able to write comments
- To be able to understand and write accessor and mutator methods
- **To be able to write methods**
- To be able to use static variables and methods
- To be able to understand variable scope
- To be able to understand and use the this keyword

Methods

- Define the objects' behavior
- Can only be called on an object that was created using the constructor
- Can return a value or not
- To call a method, you write

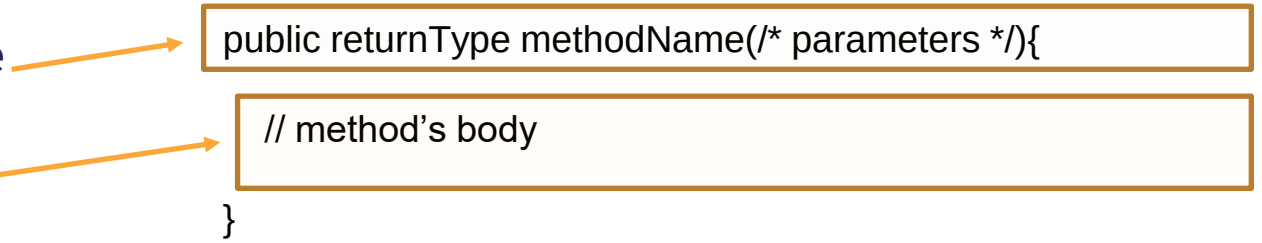
```
objectName.methodName(/* parameters or not*/);
```

Example:

```
Person p = new Person();
p.setName("Mariah");
```

Writing a method

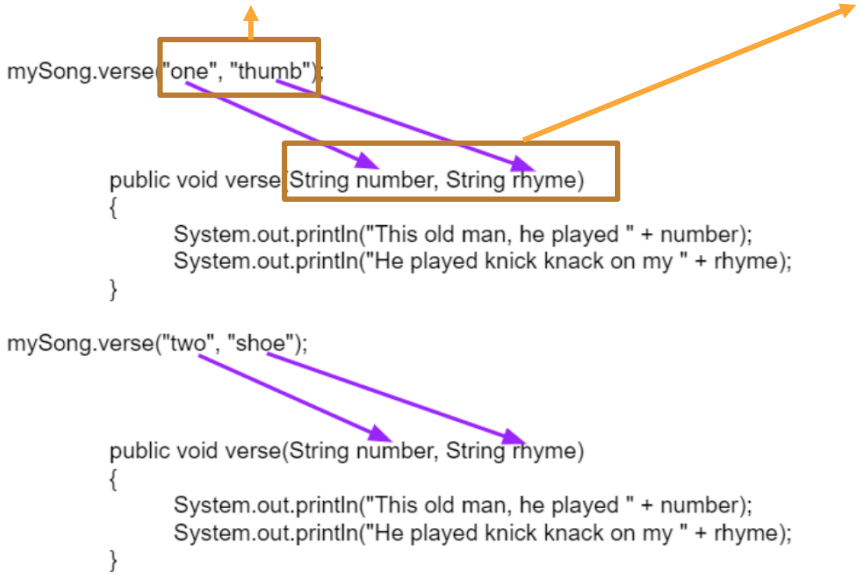
- A method has:
  - A signature
  - A body




```
public String toString() { // signature

  return "my name is" + name; // body

}
```

Methods with parameters

- When calling a method with parameters,
  you must provide **actual values** in place of the **formal parameters**



Call by value

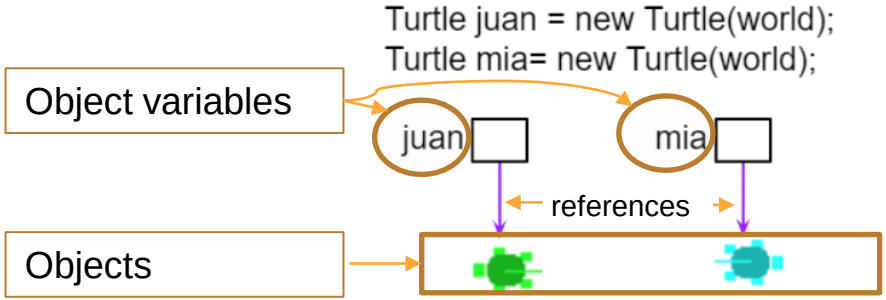
- When calling a method:
  - A copy of the value is stored in the parameter variable
  - Any **change** to the **value** inside the method is **not visible outside of the method**

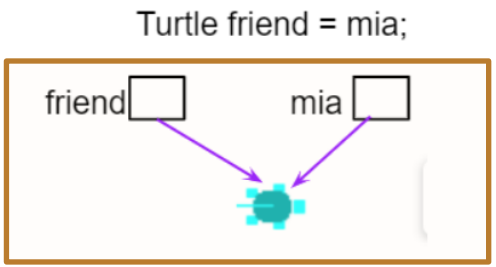
```
public class Point{
  private int  x;
  public Point(){
    x = 0;
  }
  public void setTimes2(int y){
    y = y * 2;
    x = y;
  }
}
```

```
Point p = new Point();
int z = 5;
p.setTimes2(z);
System.out.println(z) // 5
```

Call by value: Objects

- When calling a method, If you pass a value that is an **object** (String, Person, Point, etc.)
    - An **alias** (a copy of the reference) of that object is stored in the parameter variable
    - Any **change** to the **object** inside the method is **visible outside of the method**





friend and mia are **aliases**. They both reference the same object

Call by value: Objects

```
public class Point{
  private int  x;
  public Point(){
    x = 0;
  }
  public int getX(){
    return x;
  }
  public void setX(int v){
    x = v;
  }

  public void xTimes2(Point p){
    p.setX(p.getX() * 2);
  }
}
```

```
Point p1 = new Point();
Point p2 = new Point();
p2.setX(5);
System.out.println(p2.getX()); // 5

p1.xTimes2(p2);

System.out.println(p2.getX()); // 10
```

Learning Objectives

- To be able to write and use a class
- To be able to write a class constructor
- To be able to write comments
- To be able to understand and write accessor and mutator methods
- To be able to write methods
- **To be able to use static variables and methods**
- To be able to understand variable scope
- To be able to understand and use the this keyword

Static **variables and methods**

- Instance variables and methods define the attributes and behavior of the objects
- Instance variables and methods are called with the object name:
  ObjectName.methodName();
  ObjectName.variableName;

- Static variables and methods belong to the class
- Static variables and methods are called with the class (and object) name
  ClassName.staticMethodName();
  ClassName.staticVariableName;

Static **variables and methods**

- The static keyword is placed right after the public/private modifier when defining static variables and methods
- A static method can be public or private
- To declare a **static variable,** you write

public ***static*** VariableType VariableName;

Static **variables and methods**

- To define a static method, you write

```
public static MethodType MethodName(/* parameters */);
```

```
public static   int             getNumberOfStudents(){
    return numberOfStudents;
}
```

Static variables and methods

- All objects share the same copy of a static variable

```
public class Student{
  private String name;
  private int age;
  public static int numberOfStudents;

  public Student(String newName, int newAge){
    name = newName;
    age = newAge;
    numberOfStudents++;
  }


   public static int getNumberOfStudents(){
     return numberOfStudents;
   }
}
```

```
Student s1= new Student(“Alice”, 12);
System.out.print(Student.getNumberOfStudents());// 1
Student s2 = new Student(“Dwayne”, 13);
System.out.print(Student.getNumberOfStudents());// 2
Student s3 = new Student(“Rachel”, 10);
System.out.print(s3.getNumberOfStudents()); // 3
```

The same copy of *numberOfStudents* is incremented for all objects

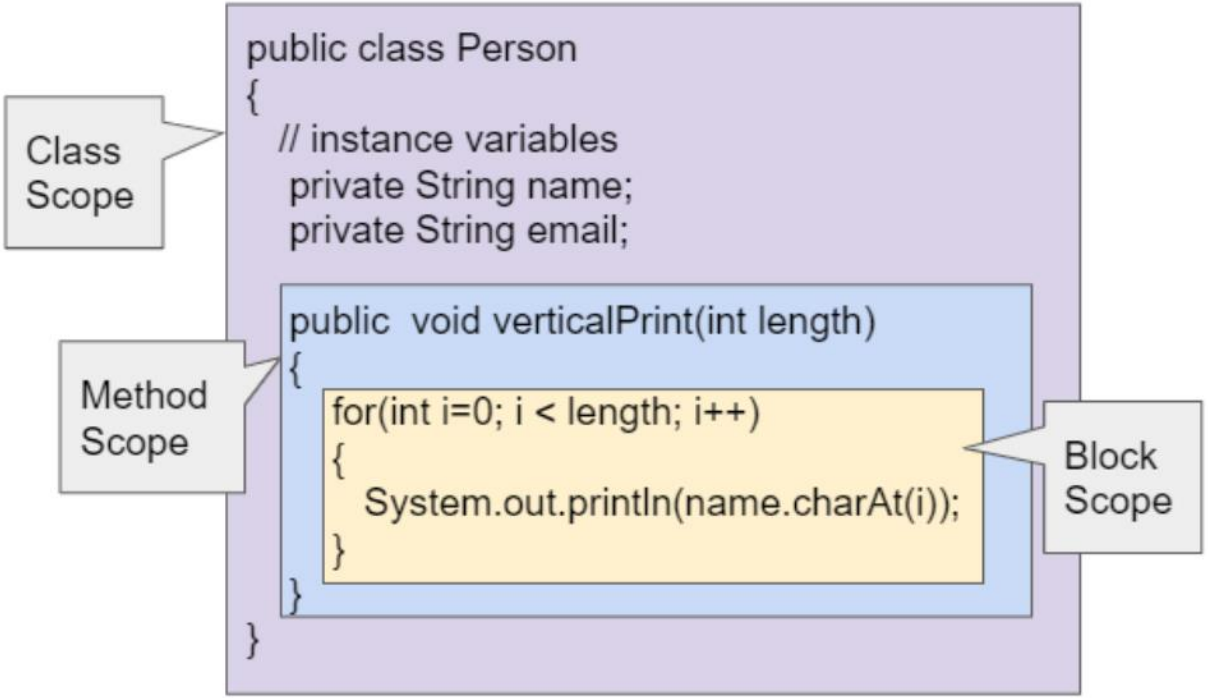
Learning Objectives

- To be able to write and use a class
- To be able to write a class constructor
- To be able to write comments
- To be able to understand and write accessor and mutator methods
- To be able to write methods
- To be able to use static variables and methods
- **To be able to understand variable scope**
- To be able to understand and use the this keyword

Variable scope

- The **scope** of a variable is where the variable is accessible
- The **scope** depends on where the variable is declared
- There are three (3) levels of scope
    - **Class level scope**: the variable is accessible in the entire class. Instance variables
    - **Method level scope**: the variable is accessible inside the method. Local variables and parameters
    - **Block level scope**: the variable is accessible inside the body of a loop. Loop control variables

Variable scope



Learning Objectives

- To be able to write and use a class
- To be able to write a class constructor
- To be able to write comments
- To be able to understand and write accessor and mutator methods
- To be able to write methods
- To be able to use static variables and methods
- To be able to understand variable scope
- **To be able to understand and use the this keyword**

this keyword

- this is used inside a non-static method to refer to the current calling object
- Can be used to refer to instance variables (with the '.' operator)
- It can be used to differentiate between instance variables and parameters

```
public class Student{
  private String name;
  private int age;

  public Student(String name, int age){
    this.name = name;
    this.age = age;
  }

   public String getName(){
     return this.name;
    }
}
```

Instance variables

parameters