Computer Science II
Handout 3

Methods and signatures

- Constructors are like methods: why does Java permit two different versions?
  - How does Java know which one needs to be called?

```
public class Rectangle {
	private double width;
	private double length;

	public Rectangle() {
		// Do nothing
	}

	public Rectangle(double l, double w) {
		// Do stuff
	}
}
```

Methods and signatures

- Does this extend to methods as well?
  - It does!

- Methods can share the *same name* using a process called *method overloading*

- Java only requires that each method has a unique *signature*, which includes:
  - The method name
  - The parameter list

Methods and signatures

- Java only cares about the *types* and their *order* in the parameter list (parameter names are irrelevant)

- Importantly, the return type is **not** a part of the method signature

Methods and signatures

- Which of the following pairs are valid method headers within the same class?

```
public void addUp(int a, int b)
public void addUp(int b, int a)

public int checkValues(boolean[] arr)
public char checkValues(boolean[] bArray)


public String concatAll(String one, int num)
public String concatAll(int num, String two)
```

Methods and signatures

- Note that overloading methods is only needed within a single class
  - Methods in *different* classes can have the same name (even the same signature!) without any problem

- Method overloading allows you to define several ways to solve the same problem

```
// Returns the sum of 'a' and 'b'
public double addValues(int a, int b)

// Returns the sum of 'a' and 'b'
public double addValues(double a, double b)
```

Methods and signatures

- Compare method signatures to determine if your overloaded method is valid

```
// Returns the sum of 'a' and 'b'
public double addValues(int a, int b)
```

```
// Returns the sum of 'a' and 'b'
public double addValues(double a, double b)
```

Method Name	Parameters
`addValues`	`(int, int)`
`addValues`	`(double, double)`
Same	*Different!*

Methods and signatures

- Which of the following groups are valid method headers within the same class?

```
public void subtract(int x, int y)
public void subBoth(int a, int b)
public void subtract(int a, double b)

public boolean validate(boolean[] arr)
public boolean validate(int[] arr)
public boolean validateAll(int[] a, int[] b)
public boolean validateFirst(int[] arr)


public void prompt(int num, double val, char a)
public void prompt(char a, int num, double val)
```

Methods and signatures

- Constructors use the same principle to perform different actions depending on which constructor variant is used

```
public class Rectangle {
	private double width;
	private double length;

	public Rectangle() {
		// Do nothing
	}

	public Rectangle(double l, double w) {
		// Initialize length and width using parameters
	}
}
```

Methods and signatures

- Remember how **`print`** and **`println`** work:

```
int a = 3;
double x = 2.5;
char c = 'q';

System.out.println(a);          // 3
System.out.print(x);            // 2.5
System.out.println(c);          // q
```

Methods and signatures

- The **`print`** and **`println`** methods highlight:
  - Use overloading when both methods perform essentially the same task with different input types
  - Use a different method name (*not* overloading) when the methods perform different tasks
  - Designing overloaded methods well makes using them much simpler!

- And always check that the signatures are actually different!

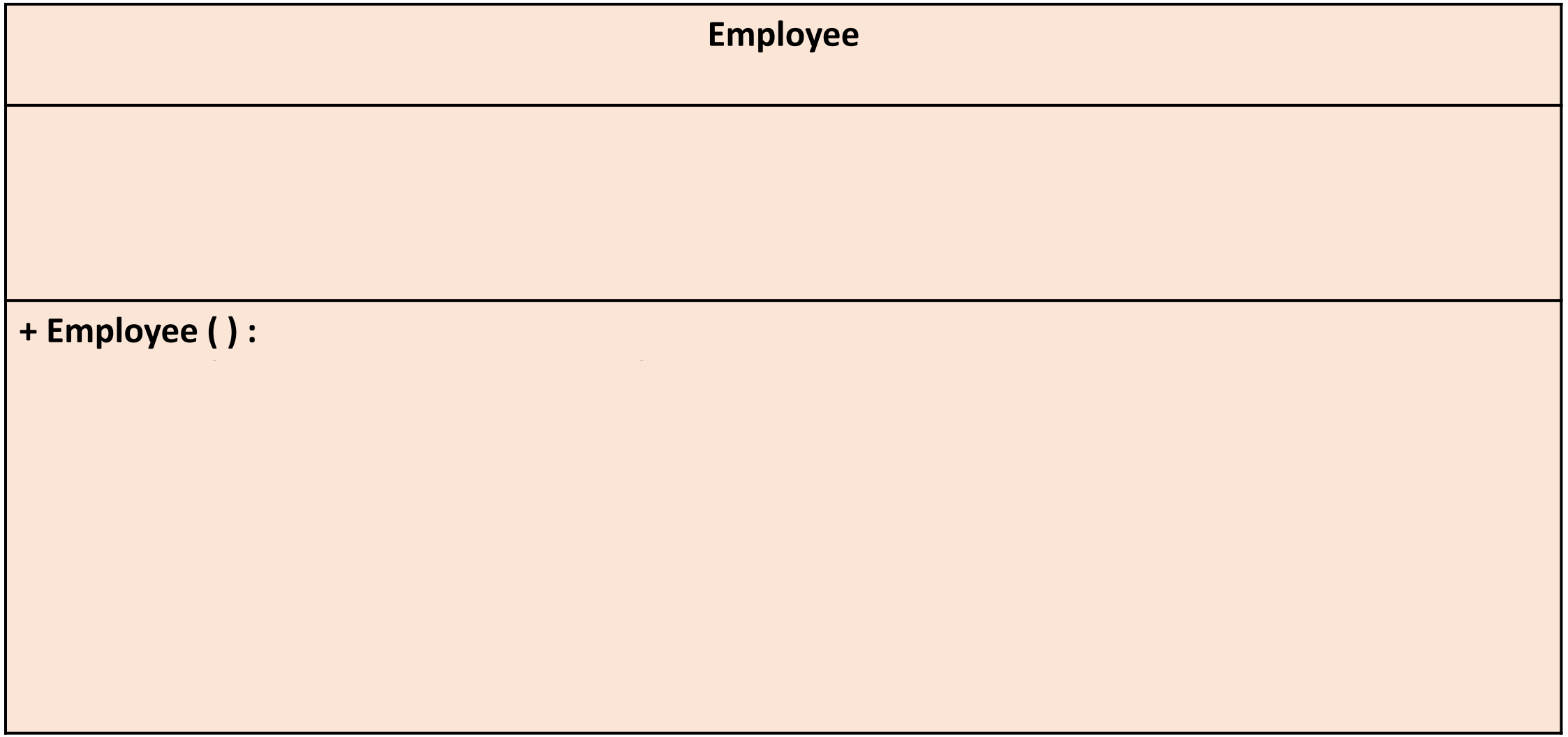
Practice with overloading methods

Design a class called **`Employee`** that tracks the name, the expected weekly hours worked, and hourly pay rate of a company employee.

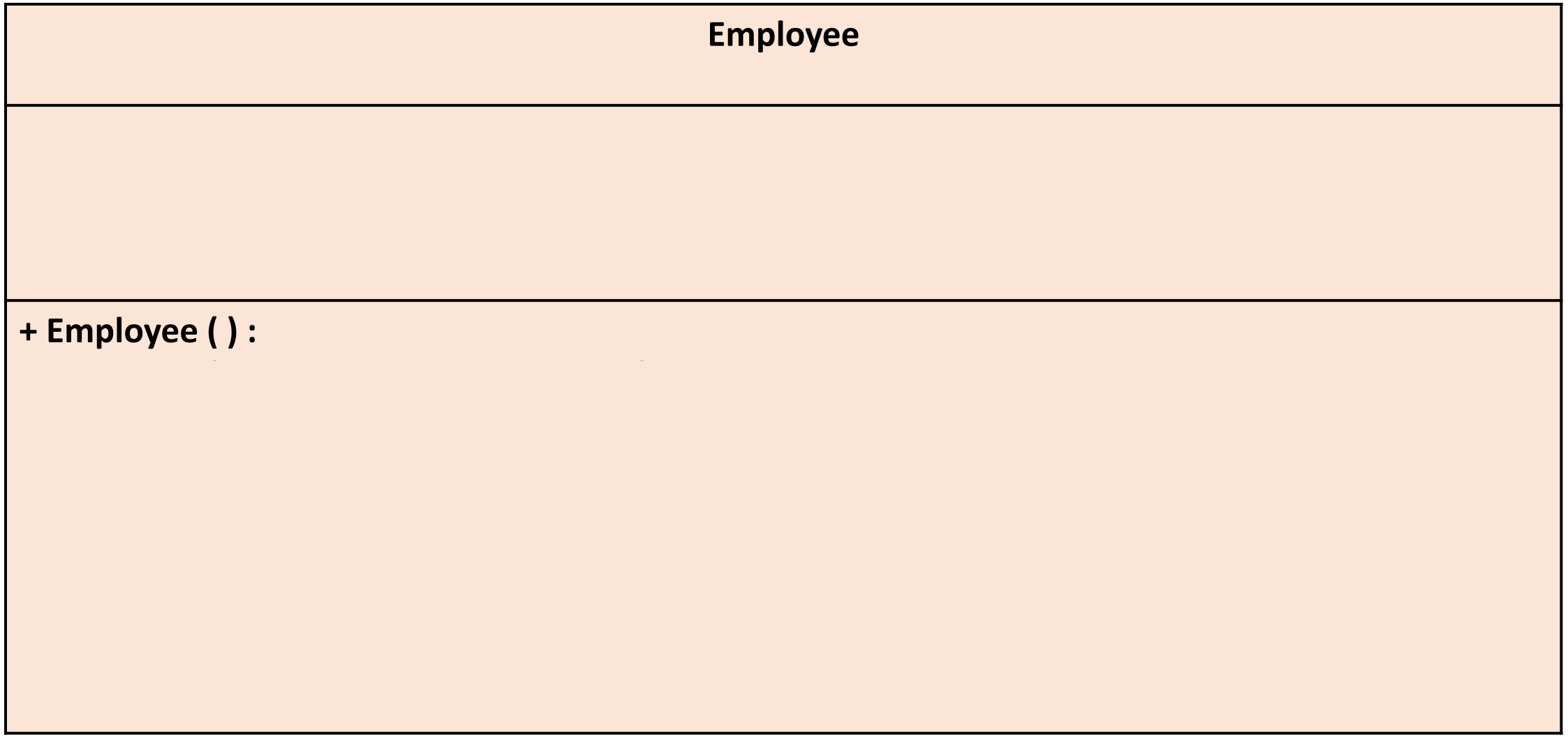
Include:

- A constructor that can set all of the attributes
- Appropriate get and set methods
- An appropriate toString method
- Methods calculateWage that will compute the amount owed to the employee:
  - If called without parameters, computes the expected weekly pay
  - If called with a single value, takes this parameter as the number of overtime hours, and computes the expected weekly pay together with 1.5 the hourly rate for the given number of overtime hours

Practice with overloading methods



Practice with overloading methods



```
public class Employee {
        private String name;
        private int hours;
        private double rate;

        public Employee()
        { }

        public void setName(String n) {
            name = n;
        }
        public void setHours(int h) {
            hours = h;
        }
        public void setRate(double r) {
            rate = r;
        }
        public String getName() {
            return name;
        }
        public int getHours() {
            return hours;
        }
        public double getRate() {
            return rate;
        }
}
```

```
public class Employee {
        private String name;
        private int hours;
        private double rate;

        public Employee()
        { }

        public void setName(String n) {
            name = n;
        }
        public void setHours(int h) {
            hours = h;
        }
        public void setRate(double r) {
            rate = r;
        }
        public String getName() {
            return name;
        }
        public int getHours() {
            return hours;
        }
        public double getRate() {
            return rate;
        }
}
```

Practice with overloading methods

Let's use this!

1. Create an **`EmployeeDemo`** class to create two new employees:
   - Jack Morrison, who works 20 hours per week at $25.50 per hour
   - Lena Oxton, who works 45 hours per week at $35 per hour

2. Print out String representations for both employees

3. Find out how much Jack is owed, given that he worked 5 hours of overtime

4. Find out how much Lena is owed, given that she did not work any overtime

Methods and signatures

```
> Jack Morrison is an employee
  Lena Oxton is an employee
  Jack Morrison earned $701.25
  Lena Oxton earned $1575.0
```

```
public class EmployeeDemo {
        public static void main(String[] args) {
            Employee emp1, emp2;
            emp1 = new Employee("Jack Morrison", 20, 25.5);
            emp2 = new Employee("Lena Oxton", 45, 35);
        }
}
```

Objects and methods

We have used Objects before as method parameters and return types

```
public static int count(double[] arr)
public static boolean[] compare(int[] a, int[] b)
```

We can also pass and return Objects that we have designed ourselves!

Objects and methods

Some possibilities for methods using Objects as parameters:

- Comparing the sizes of two Rectangles
- Comparing the balances in two BankAccounts
- Calculating the total cost of two Cars
- Finding the smallest element in two arrays
- ……

In each case, we use the attributes and operations within each Object to perform the comparison or calculation

Objects and methods – example

Add a method to **`Rectangle`** that accepts a **`Rectangle`** as a parameter and returns true if this Rectangle has a smaller area than the parameter **`Rectangle`**, false otherwise.

Designing the method:

- What is its name?
- What are the parameter types and names?
- What is the return type?
- What attributes/operations do I need from this Rectangle?

```
public                              (                    )
```

Objects and methods – example

```
public
}
```

Objects and methods – exampleDemo

Write a demo class (RectangleDemo.java) that uses the new comparison method. Create two Rectangles, initialize their attributes, and print a message indicating whether the first is larger than the second, or smaller than the second.

RectangleDemo.java

```
public class RectangleDemo {
        public static void main(String[] args) {
                Rectangle rect1, rect2;
                rect1 = new Rectangle(7.5, 18.5);
                rect2 = new Rectangle(11.3, 12.3);









        }
}
```

Objects and methods – example 2

Write a class called **`Point`** that represents a 2D point in the Cartesian plane having integer coordinates.

- The integer coordinates should be given on creation of a new **`Point`**
- Include appropriate getter, setter, and toString methods
- Include method **`isHigher`** that returns true if this point is higher (in the plane) than another point, and returns false otherwise

Objects and methods – example 2

Point
- x : int - y : int
+ Point ( ) : + Point (x : int, y : int) : + setX (x : int) : void + setY (y : int) : void + getX ( ) : int + getY ( ) : int + toString ( ) : String + isHigher (p : Point) : boolean

```
public class Point {
        private int x;
        private int y;

        // Constructors
        public Point() { }









        // Setters
        public void setX(int x) {



        public void setY(int y) {



        // Getters
        public int getX() {



        public int getY() {
```

```
        public String toString() {









        public boolean isHigher(Point p) {









}
```

```
public class Point {
        private int x;
        private int y;

        // Constructors
        public Point() { }









        // Setters
        public void setX(int x) {



        public void setY(int y) {



        // Getters
        public int getX() {



        public int getY() {
```

```
        public String toString() {









        public boolean isHigher(Point p) {









}
```

Objects and methods – example2Demo

Write a demo class (PointDemo.java) that takes input from the user to generate two points, and prints a message indicating whether the first point is higher than the second.

PointDemo.java

```
import java.util.Scanner;
public class PointDemo {
        public static void main(String[] args) {
                Point p1, p2;
                Scanner kb = new Scanner(System.in);

                System.out.print("Enter x/y co-ords for a point: ");
                p1 = new Point(kb.nextInt(), kb.nextInt());
                System.out.print("Enter x/y co-ords for another point: ");
                p2 = new Point(kb.nextInt(), kb.nextInt());









        }
}
```

Objects and methods – core attributes

In writing our toString methods, we printed out attributes that were central to the Object (along with some formatting)

Deciding on which attributes are *most* important lets us define other commonly useful methods

Objects and methods – equals

As Objects, Strings have **`.equals()`** methods that can compare two Strings

- Remember that it would generally not work to use a simple Boolean comparison!

```
String x = "Hi";
String y = "H" + "i";
boolean b = (x == y);  // ??
```

- We can write a similar method for our own Objects!
  - This should compare data that is somehow central to the Object itself

Point.java

```
public boolean equals(Point r) {




}
```

Objects and methods – copy

Copying Objects (like arrays!) also needs a dedicated method

- Remember that it *may* not work to use the regular assignment operator

```
int[] x = {2, 4, 6, 8, 10};
int[] y = x;
x[2] = 0;
boolean b = (x[2] == y[2]);      // True!
```

- Because these variables are storing *references to Objects*, the assignment operator only copies the *reference*

Objects and methods – copy

This requires a method that returns a reference to an Object

- The Object reference type is given just like any data type:

```
public int countElements(double[] arr)

public int[] getArray(int size)
public Rectangle makeRectangle()
public Circle mystery(Point p, int radius)
```

- A *reference* to these Object types is then placed on the stack
  - The Object itself remains on the heap

Point.java

```
public Point copy() {



}
```

Objects and methods – copy

Copying an Object can simply mean copying over all of the attributes

- But for the Object you designed, maybe not!

Copying an Employee could mean copying their name, hours worked, and hourly rate.

- But another common operation could be to copy everything *except* the name, to create another employee in the same position

Employee.java

```
public Employee copy() {




}

public Employee copy(String name) {




}
```