Method Overloading, Asserts, and Objects as Parameters of Methods

Colorado State University

Department of Computer Science

Slides Originally Created by Albert Lionelle (Albert.Lionelle@colostate.edu),
updated by Marcia Moraes (marcia.moraes@colostate.edu)

Announcements

TODO Reminders:

Readings are due **before** lecture

- Reading 11 (zybooks) – you should have already done that ☺
- Lab 07 – go to your lab to have your participation points
- Reading 12 (zyBooks) - you should have already done that ☺
- Lab 08 – go to your lab to have your participation points
- Reading 13 (zybooks)
- RPA 6

Keep practicing your RPAs in a spaced and mixed manner ☺

https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS_UM2jEt9rPdBktlYNT9O0orskZRG5_0eh0Q&usqp=CAU

Help Desk

Day	Time : Room
Monday	12 PM - 2 PM : CSB 120
Tuesday	6 PM - 8 PM : Teams
Wednesday	3 PM - 5 PM : CSB 120
Thursday	6 PM - 8 PM : Teams
Friday	3 PM - 5 PM : CSB 120
Saturday	12 PM - 4 PM : Teams
Sunday	12 PM - 4 PM : Teams

Recall Activity

- **Grab a paper, write your name, as it is in Canvas, and your answers to the following questions. Turn this as your attendance for today's lecture.**

- What is method overloading?
- Explain with your own words and provide examples.

Review – Method Signature

[scope] [static] TYPE name(parameters) {}

- Scope
  - public, private, protected, or blank (package protected)
  - we often use public or private
  - private is class only
  - public other classes have access
- (return) TYPE
  - Required
  - Can be any type + void.
    - Classes you create are objects, that can be returned
  - void means returns nothing
- Parameters are part of the name
  - `getArea(int x)`
  - `getArea(int x, int y)`
  - `indexOf(char x)`
  - `indexOf(char x, int start)`
- We have seen this example before
  - Called “overloading”
  - But how do we keep it DRY (**D**on’t **R**epeat **Y**ourself)?

Method Overloading

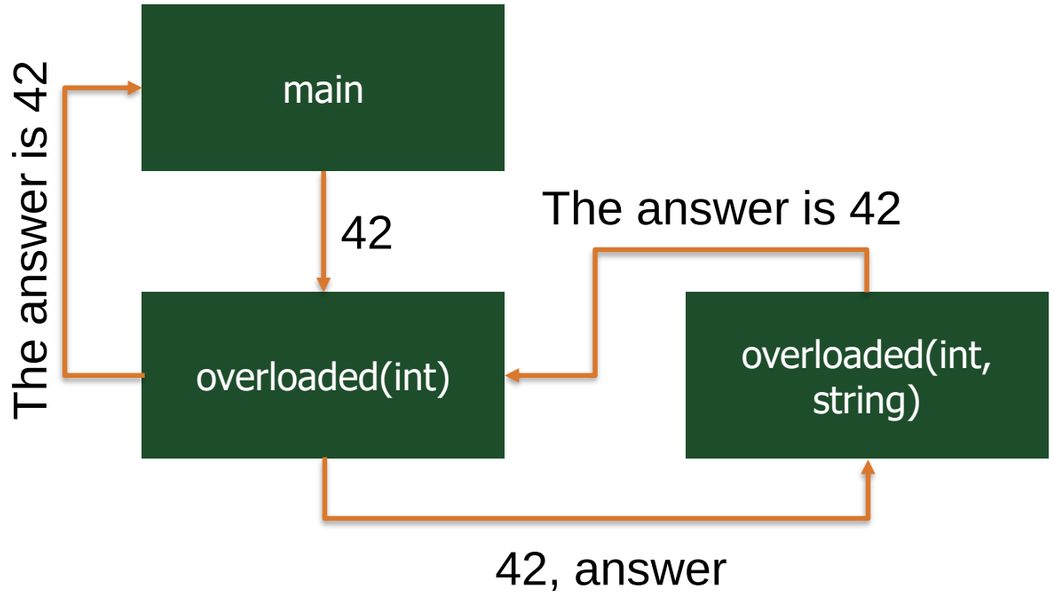
- You can have the same method name, different parameters
- Java will match the parameters on which method is called
- Best practice:
  - Methods with less parameters call the most detailed version
  - This let's you have "default" values for methods
  - Makes it so you only have one place to update!

Method Overloading

```
public static String overloaded(int x) {
    return overloaded(x, str: "Answer");
}

public static String overloaded(int x, String str) {
    return "The " + str + " is " + x;
}
```

```
public static void main(String args[]) {
    System.out.println(overloaded( x: 42));
}
```



- Method overloading works because:
  - We keep it DRY (Don't Repeat Yourself)
  - Make the more specific method do the work
  - Assume default values for different parameters
- When lost, draw it out!

Method Overloading - Practice

```
public class DatePrinter {
    public void datePrint(int day, int month, int year) {
        System.out.print("1");
    }
    public void datePrint(int day, String month, int year) {
        System.out.print("2");
        datePrint(day, Integer.parseInt(month), year);
    }
    public void datePrint(int month, int year) {
        System.out.print("3");
        datePrint(1, month, year);
        datePrint(1, String.valueOf(month), year);
    }
}
```

```
public class AppDatePrinter {
    public static void main(String args[]){
        DatePrinter dp = new DatePrinter();
        dp.datePrint(22,2, 2023);
        System.out.println();
        dp.datePrint(22,"2", 2023);
        System.out.println();
        ArrayList<String> months = new ArrayList<>();
        months.add("January"); months.add("February");
        dp.datePrint(months.indexOf("January"), 2023);
    }
}
```

What is the output?

Method Overloading - Practice

```
public class DatePrinter {
    public void datePrint(int day, int month, int year) {
        System.out.print("1");
    }
    public void datePrint(int day, String month, int year) {
        System.out.print("2");
        datePrint(day, Integer.parseInt(month), year);
    }
    public void datePrint(int month, int year) {
        System.out.print("3");
        datePrint(1, month, year);
        datePrint(1, String.valueOf(month), year);
    }
}
```

```
public class AppDatePrinter {
    public static void main(String args[]){
        DatePrinter dp = new DatePrinter();
        dp.datePrint(22,2, 2023);
        System.out.println();
        dp.datePrint(22,"2", 2023);
        System.out.println();
        ArrayList<String> months = new ArrayList<>();
        months.add("January"); months.add("February");
        dp.datePrint(months.indexOf("January"), 2023);
    }
}
```

What is the output?

1

21

3121

Method Overloading – Coding Along

```
import java.util.ArrayList;
public class Contact {
   private long phone;
   private String name;
   public Contact(String name) {
      this.name = name;
      phone = 0;
   }
   public Contact(String name, String phone){
      //to do
   }
   public Contact(String name, long phone){
      //to do
   }
```

```
public String getFormattedPhone() {
   return String.format("(%d) %d-%d", getAreaCode(), getPrefix(), getNumber());
}
public int getPrefix() {
   long tmp = phone / 10000;
   return (int) (tmp % 1000);
}
public int getAreaCode() {
   long tmp = phone / 10000000;
   return (int) (tmp % 1000);
}
public int getNumber() {
   return (int) phone % 10000;
}
public long getPhone() {
   return phone;
}
```

Method Overloading – Coding Along

```
public void setPhone(long phone) {
   // to do
}
public void setPhone(String phone) {
   // to do
  //transform the String into a long number
  //remove all characters that are not numbers
  //Long.parseLong(string)
 //call overloaded setPhone
}
public String toString() {
   return String.format("Name: %s, phone: %s", name,
getFormattedPhone());
}
```

```
public static void main(String[] args) {
      ArrayList<Contact> advisors = new ArrayList<>();
      Contact bess = new Contact("Bess");
      bess.setPhone(9704915944L); // the "L" at the end is how
we tell java it is a long number
      advisors.add(new Contact("Gabbi","(970) 491-3739"));
      advisors.add(new Contact("Tran"));
      advisors.add(new Contact("Heidi"));
      advisors.add(bess); // just doing this so you can see
adding other ways to add objects to ArrayLists
      for(Contact advisor:advisors) {
         if(advisor.getPhone() > 0) {
            System.out.println(advisor);
         }
      }
   }
}
```

Reminder: Keep It Simple

- Methods are the conquer: divide -> conquer -> glue
  - Which means, the smaller problem to solve, the better
  - Keep what you do in a method simple
  - If you write 20 lines, you probably have written too much
  - If you cut and paste, you need a method.
- Turn problems into questions
  - **What is your quest?**
  - What do you know
  - What do you need (parameters)

Parameters In More Depth

Let's Talk About Variables and Memory

- Discuss: Given the following program

Program Start:

- What is the value of:
  - w
  - l
  - rectange.length
  - rectangle.width

Program End?

- What is the value of them?

```
public class YourProgram {
   public static void modifyValues(Rectangle rectangle, int w, int l) {
      rectangle.setLength(rectangle.getLength() / 2);
      rectangle.setWidth(rectangle.getWidth() * 3 + 1 );
      w = w * 3 + 1;
      l = l  / 2;
   }
   public static void printValues(Rectangle rectangle, int w, int l) {
      System.out.printf("Rectangle: Width %d, Length: %d, Area: %d%n",
            rectangle.getWidth(), rectangle.getLength(), rectangle.getArea());
      System.out.printf("Values of w: %d, of l: %d%n", w, l);
      System.out.printf("Width == w? %b,  Length == l? %b%n%n",
            rectangle.getWidth() == w, rectangle.getLength()==l);
   }
   public static void main(String[] args) {
      int w = 5;
      int l = 10;
      Rectangle rectangle = new Rectangle(w,l);
      printValues(rectangle, w, l);
      modifyValues(rectangle, w, l);
      System.out.println("Values Modified!");
      printValues(rectangle, w, l);
   }
}
```

```
public class Rectangle {
   private int length = 0;
   private int width = 0;

   public Rectangle(int width, int length) {
      setWidth(width);
      setLength(length);
   }

   public void setLength(int length) {
      if(length>0) this.length = length;
   }

   public void setWidth(int width) {
      if(width > 0) this.width = width;
   }

   public int getLength() {
      return length;
   }

   public int getWidth() {
      return width;
   }

   public int getArea() {
      return length*width;
   }
}
```

The Output From the Program….

- The values in Rectangle were changed!
- The values of w, l - where not changed!

- What happened?
- Why did this happen?
- Let’s look at memory

```
Rectangle: Width 5, Length: 10, Area: 50
Values of w: 5, of l: 10
Width == w? true,  Length == l? true

Values Modified!
Rectangle: Width 16, Length: 5, Area: 80
Values of w: 5, of l: 10
Width == w? false,  Length == l? false
```

Java doesn't 'copy' objects

Digging Deeper:
This is called the memory stack, explored in later courses (165/270)

- Here is the memory of the program (simplified)

```
int w = 5;
int l = 10;
Rectangle rectangle = new Rectangle(w,l);

modifyValues(rectangle, w, l);

Now running the modifyValues method
modifyValues(Rectangle rectangle, int w, int l)

rectangle.setLength(rectangle.getLength() / 2);
rectangle.setWidth(rectangle.getWidth() * 3 + 1 );
w = w * 3 + 1;
l = l  / 2;
```

Variable list for main!

10
x873
5

Variable list for modifyValues!

x873
5 16
10 5

x873

w = 5 16
l = 10 5

modifyValues finishes (local variables go away), returns to **main**

Passing Objects as Parameters

```
public static int getAndCountNameLetters(Scanner scnr) {
    String name = "";
    if (scnr.hasNext()) {
        name = scnr.next();
    }
    return name.length();
}

public static void main(String[] args) {
    int firstNameLetterCount;
    int lastNameLetterCount;
    Scanner scnr = new Scanner(System.in);
    System.out.println("Enter a person's first and last names:");
    firstNameLetterCount = getAndCountNameLetters(scnr);
    lastNameLetterCount = getAndCountNameLetters(scnr);
    System.out.println("The first name has " + firstNameLetterCount + " letters.");
    System.out.println("The last name has " + lastNameLetterCount + " letters.");
}
```

Scanner object as a parameter in method definition

Creates scnr object of Scanner class type

Sending an object as a parameter

Practice – Objects as Parameters

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

    public Rectangle(double height, double width){
        setHeight(height);
        this.setWidth(width);
    }
    public void setHeight(double height){
        this.height = height ;
    }
    public void setWidth(double width){
        this.width = width;
    }
    public double getHeight(){
        return height;
    }
    public double getWidth(){
        return width;
    }
}
```

- Rectangle Class
- Write a method that calculates the area of a rectangle.
- Write a method that receives a Rectangle as a parameter, compare the areas and returns a String with the height, width, and area of the biggest rectangle. If it is the same area, return a String informing that.
- RectangleApplication Class
- Write a RectangleApplication class that has the following methods:
  - readDouble: receives a Scanner as a parameter, read, and return an double value;
  - main: creates two Rectangle objects (use readDouble to read the values to create the objects), prints both objects height and width, calls the methods that return the biggest rectangle.