Method Overloading

In this lecture we will focus on methods, and more importantly method overloading.

Your future in CS

I used to include this on my slides, but since these slides have changed - going to just leave it up here for every notebook. I get a lot of questions about more programming courses, the concentrations, and minors in computer science. Here is a brief reminder.

CS 165 – Next Course In Sequence, also consider CS 220 (math and stats especially)

- CO Jobs Report 2021 – 77% of *all* new jobs in Colorado require programming
- 60% of all STEM jobs requires *advanced* (200-300 level)
- 31% of all Bachelor of Arts degree titled jobs also required coding skills
- 2016 Report found on average jobs that require coding skills paid $22,000 more

- Concentrations in CS:
    - Computer science has a number of concentrations. General concentration is the most flexible, and even allows students to double major or minor pretty easily. The others are specialized paths in CS. Software Engineering, Computing Systems, Human Centered Computing, Networks and Security, Artificial Intelligence, and Computer Science Education.
    - Minors: We have three minors. Minor in Computer Science (choose your own adventure minor), Minor in Machine Learning (popular with stats/math, and engineering), and Minor in Bioinformatics (Biology+Computer Science)

Reading Check-in Question:

Given the following code, what is printed. Code is slightly modified from 13.2.1 method naming overloading activity.

In [13]:
```
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);
    }
}

DatePrinter printer = new DatePrinter();
```

In [5]:
```
printer.datePrint(3, 7, 2022);
```

```
1
```

In [15]:
```
printer.datePrint(1, "7", 2022);
```

```
21
```

In [11]:
```
ArrayList<String> months = new ArrayList();
months.add("July"); months.add("August");
printer.datePrint(months.indexOf("July"), 2022);
```

```
3121
```

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!

Let's look at code.

In [17]:
```
public static String overloaded(int x) {
    return overloaded(x, "answer");
}

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


System.out.println(overloaded(42));
```

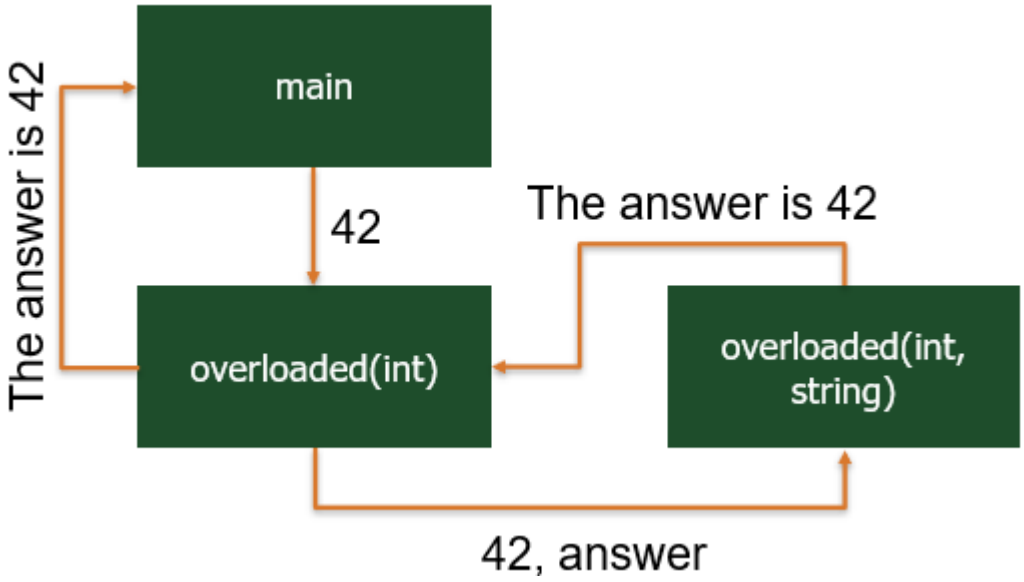
```
The answer is 42
```

Tracing methods

Box drawing style

- Draw a box for each method
- Have a line going from method to method
  - in order of the call.
  - include parameters
- Have a line going back to the calling method

- include the return value



Indentation Style

You can also write it out, often with indents.

- In Main -> Calling overloaded(42)
  - in overloaded(42)
  - calling overloaded(42, "answer")
    - in overloaded(42)
    - returning "The answer is 42"
  - returning "The answer is 42"
- Printing "The answer is 42"
- end program.

In class activity

For this In class activity, you will write both overloaded constructors and overloaded methods.

In Contact.java, you will find a number of methods implemented for you. Mainly dealing with parts of getting a phone number. We need you to implement:

- Constructors:
  - Contact(String name, String phone)
  - Contact(String name, long phone)
- Methods:
  - public void setPhone(long phone)
  - public void setPhone(String phone)

public void setPhone(String phone)

This is the hardest one building off of last Friday. This method should take in a formatted number, and remove anything that isn't a number. This means using the following tools:

- Character.isDigit()
- .charAt(int) - from String, so phone.charAt(int)
- a for loop, that loops from 0 -> phone.length()
- Long.parseLong(String) - to convert the final number only number to a long
- Then call setPhone(long)

In [30]:
```
public class Contact {
    long phone = 0;
    String name;

    public Contact(String name) {
        this.name = name;
    }

    public Contact(String name, String phone) {
        this.name = name;
        setPhone(phone);
    }

    public Contact(String name, long phone) {
        this.name = name;
        setPhone(phone);
    }


    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;
    }

    public void setPhone(long phone) {
        this.phone = phone;
    }

    public void setPhone(String phone) {
        String tmp = "";
        for(int i = 0; i < phone.length(); i++) {
```

```
            if(Character.isDigit(phone.charAt(i))) {
                tmp += phone.charAt(i);
            }
        }
        setPhone(Long.parseLong(tmp));
    }

    public String toString() {
        return String.format("Name: %s, phone: %s", name, getFormattedPhone());
    }

}

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, ar

advisors.add(new Contact("Elisa", "(970) 491-3739"));
advisors.add(new Contact("Tran"));
advisors.add(new Contact("Heidi"));
advisors.add(bess);

for(Contact advisor : advisors) {
    if(advisor.getPhone() > 0) {
        System.out.println(advisor);
    }
}
```

```
Name: Elisa, phone: (970) 491-9147
Name: Bess, phone: (970) 491-1352
```

Iterative Development

- Programs should be developed in:
  - Small tests
  - Constantly compiling
  - Constantly testing
  - Even only a few lines at a time!
- This is called iterative development
- Has been shown to be better than writing a bunch at once!

Methods for functional programmers?

Let's talk about math functions.

- $f(x) = 2^x$
- $f(2) = 2^2 = 4$
- $f(3) = 2^3 = 8$
- $f(5) = 2^5 = 32$
- $f(y, x) = y^x$

The power of a function is to repeat and reuse, and return a value.

Such a setup is easier to:

- debug
- deal with concurrency
- deal with speed boosts
- easier to swap implementations

As you write methods, really focus on this idea!

- What is my quest? (task)
  - What are the tasks I need to complete towards my tasks?
- What do I know?
- What do I need?

Above all, continue to go back to

- Divide
  - Break tasks into smaller parts
  - Solve those tasks - hint if you are writing more than 20 lines, should probably beak it up into smaller parts
- Conquer
- Glue