Lecture 2

Exam Questions

- Make sure that you've signed up for a time in Canvas
- Show up on time. Bring your ID & pencil.
- You have 50 minutes to finish the test.
- Quiz is open to study from until tomorrow.

Pass by Value

The call

```
areaRec(5, 7);
```

in

```
doRectangularShapes()
```

passes the integer values 5 and 7 to `areaRec`.

This will become relevant later in the course

Overloading

Notice that there are e.g. two methods volumeBlck, with two different method signatures:

```
public int volumeBlck(int length, int width, int height)
```

and

```
public static int volumeBlck(int width)
```

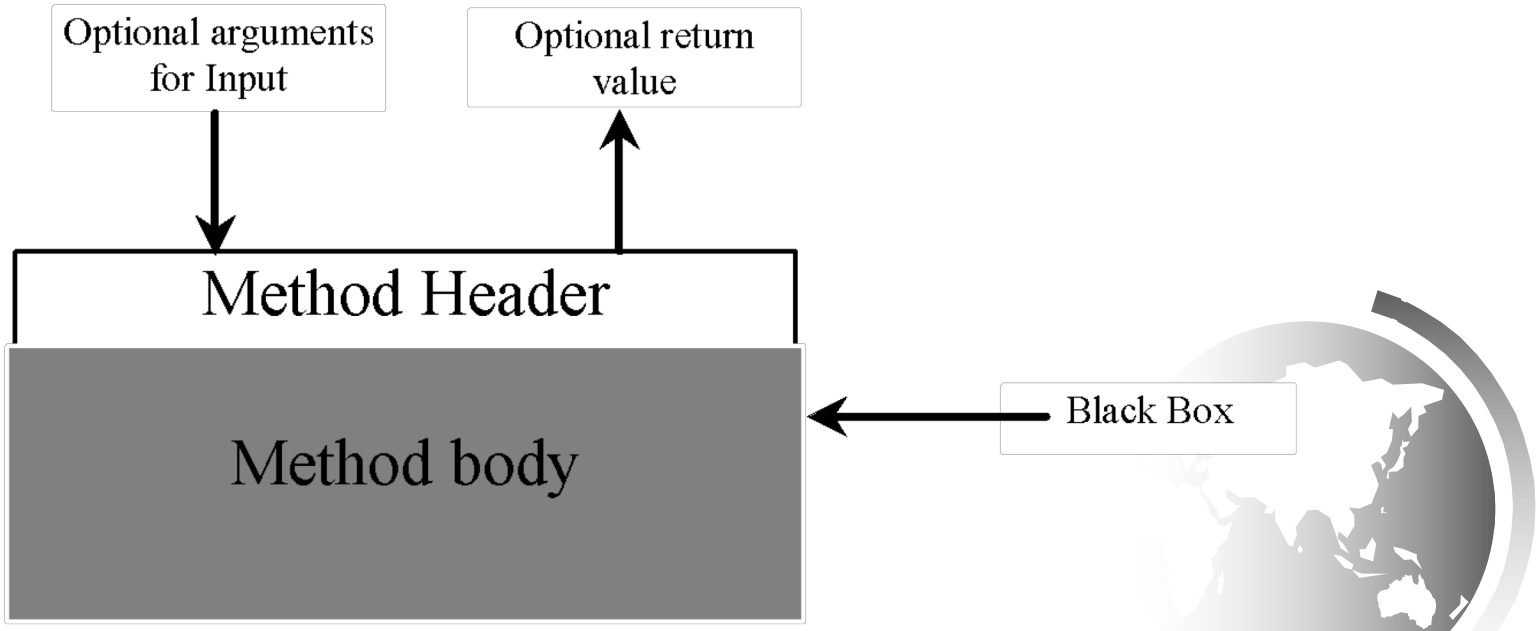
We call this method overloading. A call will check the number and types of the parameters and select the method with the matching method signature.

E.g. `volumeBlck(11)` will select

```
public static int volumeBlck(int width)
```

Method Abstraction

You can think of the method body as a black box that contains the detailed implementation for the method.



Benefits of Methods

- Write a method once and reuse it anywhere.
- Hide the implementation from the user.
- Reduce complexity (e.g. of main), thereby increasing the readability of your program.
- Simplify maintenance: if the method needs to change, you only change it in one place.

  (and the user does not need to know about it)

Your Turn!

Write two **methods** that will calculate the perimeter of a rectangle and triangle

```
public int perimeter(int length, int width)
```

and

```
public int perimeter(int a, int b, int c)
```

Introduction to Interfaces

Interfaces - motivation

- Consider the task of writing classes to represent 2D shapes such as `Ellipse, Circle, Rectangle and Square`. There are certain attributes or operations that are common to all shapes: e.g. their area

- Idea of interface: contract:

  "I'm certified as a 2D shape. That means you can be sure that my area can be computed."

Interfaces

```
interface <interface_name>
{
// declare constant fields
// declare methods that abstract
// by default.
}
```

Interfaces

```
interface 2DShape
{
int areaCircle(int);//circle
int areaSquare(int, int); //square
}
```

Interfaces

- **interface**: A list of methods that a class promises to implement.
  - Only method **stubs** (method without a body) and constant declarations in the interface, e.g.

```
public double PI =  3.14159;

public int areaRec(int length, int width);
```

  - A class **can implement** an interface
    - A rectangle has an area that can be computed by the method AreaRec
    - If a class implements an interface, it must have methods for all methods stubs in the interface.

Implementing an interface

- A class can declare that it *implements* an interface:

```
public class <name> implements <interface name> {
     ...
}
```

- This means the class needs to contain an implementation for each of the methods in that interface.

  (Otherwise, the class will fail to compile.)

Your Turn!

You wrote two methods that calculate the perimeter of a rectangle and a triangle

```
public int perimeter(int length, int width)
```

and

```
public int perimeter(int a, int b, int c)
```

How could you implement an interface for these methods?