CSC207 - Review of key concepts

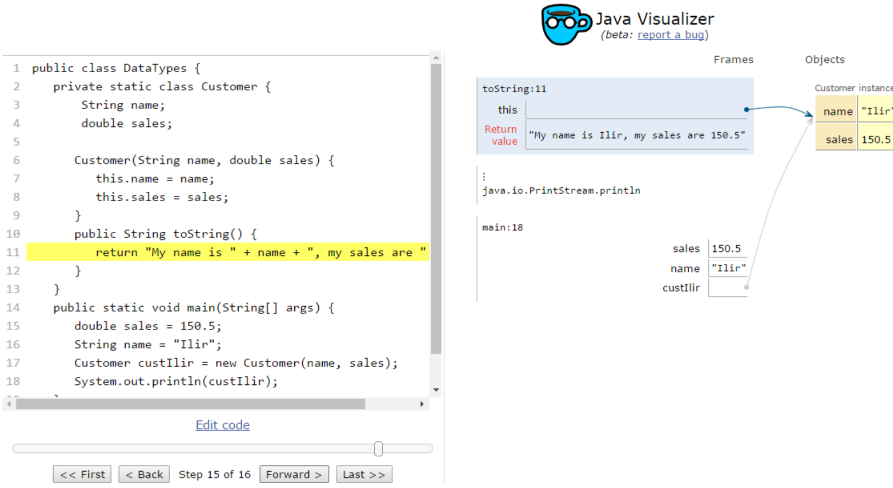
Ilir Dema

Summer 2016

Review Topics

1. Variables and data types
2. Access modifiers
3. Methods and parameter passing
4. Inheritance and abstract classes
5. Collections and generics
6. Interfaces

A sample Java program



Java Visualizer
(*beta:* report a bug)

```
public class DataTypes {
   private static class Customer {
       String name;
       double sales;

      Customer(String name, double sales) {
         this.name = name;
         this.sales = sales;
      }
      public String toString() {
         return "My name is " + name + ", my sales are "
      }
   }
   public static void main(String[] args) {
      double sales = 150.5;
      String name = "Ilir";
      Customer custIlir = new Customer(name, sales);
      System.out.println(custIlir);
```

Frames

```
toString:11
  this
  Return value  "My name is Ilir, my sales are 150.5"

:
java.io.PrintStream.println

main:18
  sales     150.5
  name      "Ilir"
  custIlir
```

Objects

Customer instance

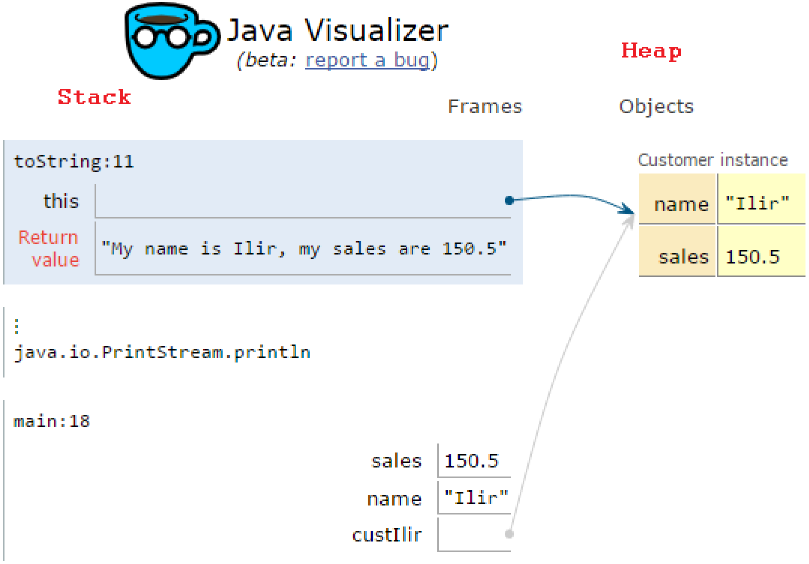
name	"Ilir"
sales	150.5

Variables and data types

Based on the data type, a variable can belong to a:

- Primitive Data Type:
  - `boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double`
  - A primitive variable is stored on the frame stack of the current thread (process)
  - That means the cell referred by variable name contains the actual value of the variable
- Reference Data Type:
  - `Customer` (see previous slide - `Customer` is a custom data type
  - Data type declared in some Java package (`Scanner`, `JFrame,...`)
  - The cell referred by the reference variable contains the address of some object on the heap.

Java memory model



```
toString:11
        this
Return value   "My name is Ilir, my sales are 150.5"

⋮
java.io.PrintStream.println

main:18
        sales    150.5
        name     "Ilir"
        custIlir
```

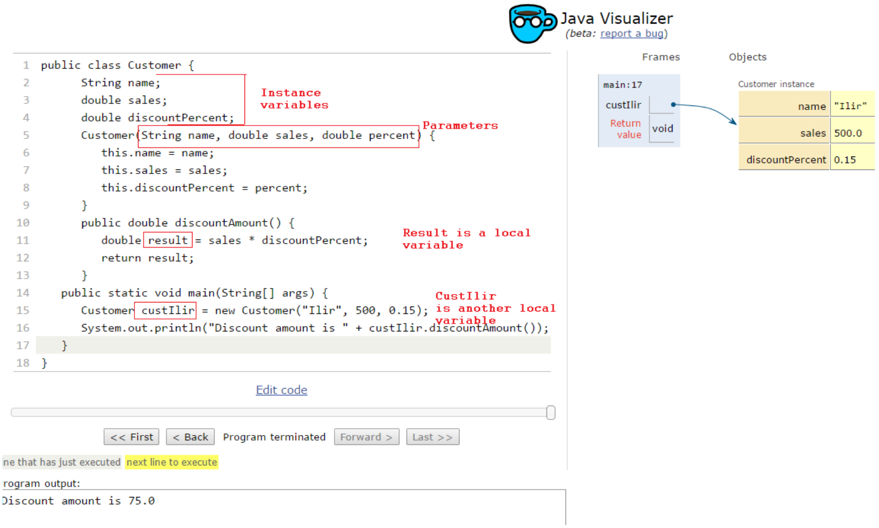
name	"Ilir"
sales	150.5

Instance variables, local variables, constants

- Based on their location on the code, a variable can be:
  - Instance (class) variables
  - Local variables
  - Parameters (also local variables)
- Paremeters and local variables are visible within the segment of code where they are declared.
- Class (instance) variables visibility can be changed using access modifiers.
- Constants do not change throughout the life of the program. They can be declared using the `final` keyword:

```
public final double NUMBER_PI = 3.14;
```

Instance variables, parameters, local variables

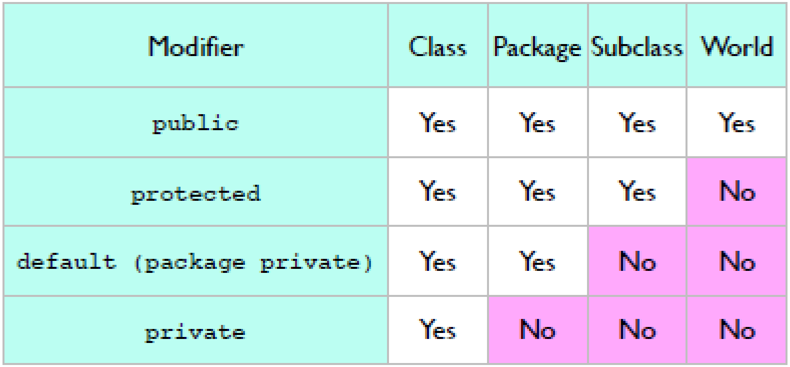


```
public class Customer {
     String name;
     double sales;
     double discountPercent;
     Customer(String name, double sales, double percent) {
        this.name = name;
        this.sales = sales;
        this.discountPercent = percent;
     }
     public double discountAmount() {
        double result = sales * discountPercent;
        return result;
     }
  public static void main(String[] args) {
     Customer custIlir = new Customer("Ilir", 500, 0.15);
     System.out.println("Discount amount is " + custIlir.discountAmount());
  }
}
```

```
Discount amount is 75.0
```

Access Modifiers

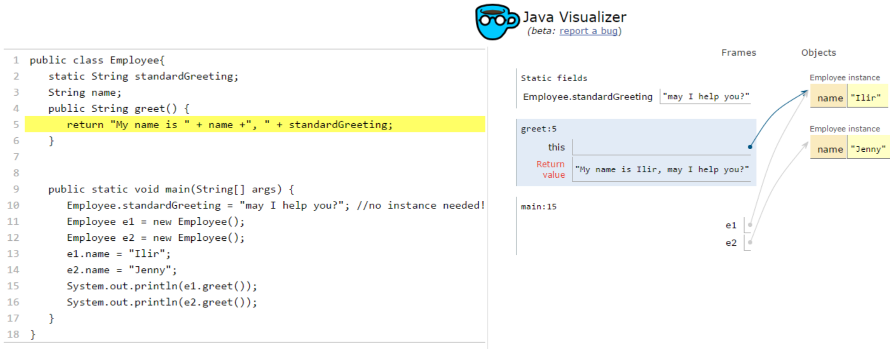
- Class members (variables/methods) can be declared public, protected, package-protected, or private.



Modifier	Class	Package	Subclass	World
`public`	Yes	Yes	Yes	Yes
`protected`	Yes	Yes	Yes	No
`default (package private)`	Yes	Yes	No	No
`private`	Yes	No	No	No

Static modifier

`static` modifier is used to declare a class variable or method that does not depend/need an instance of the class. It can be combined with access modifiers as needed. Also it is a good idea to make the constants static.



```java
public class Employee{
    static String standardGreeting;
    String name;
    public String greet() {
        return "My name is " + name +", " + standardGreeting;
    }


    public static void main(String[] args) {
        Employee.standardGreeting = "may I help you?"; //no instance needed!
        Employee e1 = new Employee();
        Employee e2 = new Employee();
        e1.name = "Ilir";
        e2.name = "Jenny";
        System.out.println(e1.greet());
        System.out.println(e2.greet());
    }
}
```

Static fields
Employee.standardGreeting "may I help you?"

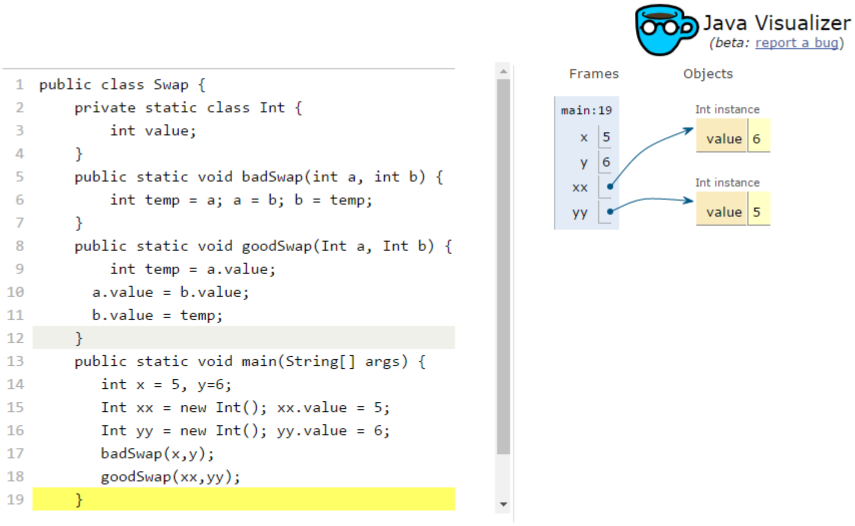
greet:5
this
Return value "My name is Ilir, may I help you?"

main:15
e1
e2

Employee instance
name "Ilir"

Employee instance
name "Jenny"

Passing of parameters by value versus by reference



```
public class Swap {
    private static class Int {
        int value;
    }
    public static void badSwap(int a, int b) {
        int temp = a; a = b; b = temp;
    }
    public static void goodSwap(Int a, Int b) {
        int temp = a.value;
      a.value = b.value;
      b.value = temp;
    }
    public static void main(String[] args) {
       int x = 5, y=6;
       Int xx = new Int(); xx.value = 5;
       Int yy = new Int(); yy.value = 6;
       badSwap(x,y);
       goodSwap(xx,yy);
    }
```

Inheritance

- Inheritance allows one class to inherit the data and the methods of another class.
- In a subclass, `super` refers to the part of the object defined by the parent class.
  - Use `super.‘‘attribute’’` to refer an attribute (data member of method) in the parent class.
  - Use `super(‘‘arguments’’)` to call a constructor defined in the parent class.
- If the constructor of the parent class is intended to be called, the `super(‘‘arguments’’)` must be the first line of code of the constructor.
  - Otherwise the default (no argument) constructor in the parent class is called.

Collections and Generics

- Often is useful to have implementations of certain Abstract Data Types (ADT).
- It is desirable that the implementation of the ADTs be independent of the data type stored in the chosen structure.
- This can be achieved through so called generics - a way of extending static typing to classes when the exact type of data the classes will operate on is unknown.
- For example, we may be interested to create Lists of Strings, Points (recall example from last lecture), etc.
- A type enclosed within angle brackets, for example `ArrayList<T>` means the programmer should replace `T` with the desired data type.
- Example: `ArrayList<Point> polygon = new ArrayList<Point>();`
- Collections are objects that hold other objects.

Java Interfaces

- A java Interface is similar to a Java class
  - can include variable declarations
  - can include methods
- However
  - Variables must be constants
  - Methods must be abstract.
- A Java interface cannot be instantiated.
- Apart from applications that we have seen, an interface can also be used to decouple certain operations from their implementation.

The `Comparable` interface

- Often it is desirable to establish an ordering of elements in a class.
- An ordering relation can be established through a compararison operation which must satisfy:
  - Every two element must be comparable. In addition, the comparaison must be:
  - Reflexive: $\forall a : a \leq a$
  - Antisymmetric: $\forall a, b : a \leq b \wedge b \leq a \implies a = b$
  - Transitive: $\forall a, b, c : a \leq b \wedge b \leq c \implies a \leq c$.
- Java offers the `Comparable<T>` interface that has a single method:

  public int compareTo(T other) which must return:
    - A negative integer if `this` less than `other`
    - Zero of `this` equals `other`
    - A positive integer if `this` greater than `other`