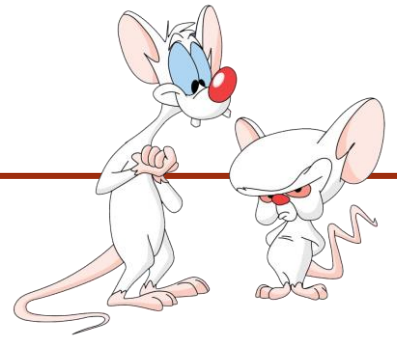
COMP 250

INTRODUCTION TO COMPUTER SCIENCE

Week 3-4 : OOD9

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Java interface Comparable

JAVA `Comparable` INTERFACE

- The Java `Comparable` interface is used to define an ordering on objects of user-defined class.
- Why would you want that? Well, if y ist of objects from a given class you might want to be ab
- `Comparable` is part of `java.lang` package and contains only one method named `compareTo(Object)`.

JAVA Comparable INTERFACE


```
public i                                T>{
    int c
}
```


https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html

JAVA `Comparable` INTERFACE

Some of the methods from certain Java classes use `compareTo()` in their implementation. To function correctly, they assume to be working with `Comparable` generic types. Examples:

- **`sort()` from `Arrays`.**

JAVA `Comparable` INTERFACE

Some of the methods from certain Java classes use `compareTo()` in their implementation. To function correctly, they assume to be working with `Comparable` generic types. Examples:

- **`sort()` from `Collectio`**

String IMPLEMENTS Comparable

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

CLASSES THAT IMPLEMENT `Comparable`

- `Character`, `Integer`, `Float`, `Double`, `BigInteger`, **etc. all implement** `Comp`

- **You cannot compare objects of ses using the "<" operator. Instead use** `compareTo()`.

HOW TO IMPLEMENT `Comparable`

- **Add `implements Comparable` in the definition of the class.**

- **Implement `compa`ass.**


```
public class T implements Comparable<T>{
   public int compareTo(T o) {...}
}
```

REQUIREMENT FOR IMPLEMENTING `compareTo()`

Consider two variable `t1` and `t2` or type `T`. Then,

`t1.compareTo(t2` $\begin{cases} \text{int} & \text{, if } t1 < t2 \\ & \text{, if } t1 = t2 \\ & \text{, if } t1 > t2 \end{cases}$

The relation should also be anticommutative and transitive.

Highly recommended:

```
(t1.compareTo(t2)==0) == (t1.equals(t2))
```

EXAMPLE - CIRCLE

- Sometimes deciding how to compare elements of a given type can be straightforward.
- Let's think about the data


```
public class Circ
   private double
   :
}
```

- How should we implement `compareTo()` and `equals()` in order to establish a *natural ordering* between elements of type `Circle`?

EXAMPLE - CIRCLE

- **How should we implement `compareTo()` and `equals()` in order to establish a *natural ordering* between elements of type `Circle`?**

We could simply compare their radius (or their area).

CIRCLE – compareTo()

```
public class Circle extends Shape implements Comparable<Circle>{
   private double radius = 5;

   public int compareTo(Circle c) {
      if (this.radius < c.r
         return -1;
      else if (this.radius
         return 0;
      else
         return 1;
   }

   public boolean equals(Object obj) {
      return obj instanceof Circle && ((Circle) obj).radius == this.radius;
   }
}
```

EXAMPLE – ORC

- Other times, is not so straightforward. Suppose we have created a new data type `Orc`.
- How should we compare type?

Base on their name? On their height? On their weapon? On who is scarier?

ORC – compareTo() TAKE 1

```
public class Orc implements Comparable<Orc> {
   private String name;
   private int height;
   private Weapon w;
   public int compareTo(O
      if(this.height < o.h
         return -1;
      } else if(this.height == o.heigh
         return 0;
      }  else {
         return 1;
      }
   }
}
```


- Note that in this case we probably don't want to consider two Orcs with the same height to be equal.
- This implies that the implementation of compareTo() violates the Java API recommendations.
- Such violation should be clearly indicated using the following language: "Note: this class has a natural ordering that is inconsistent with equals."

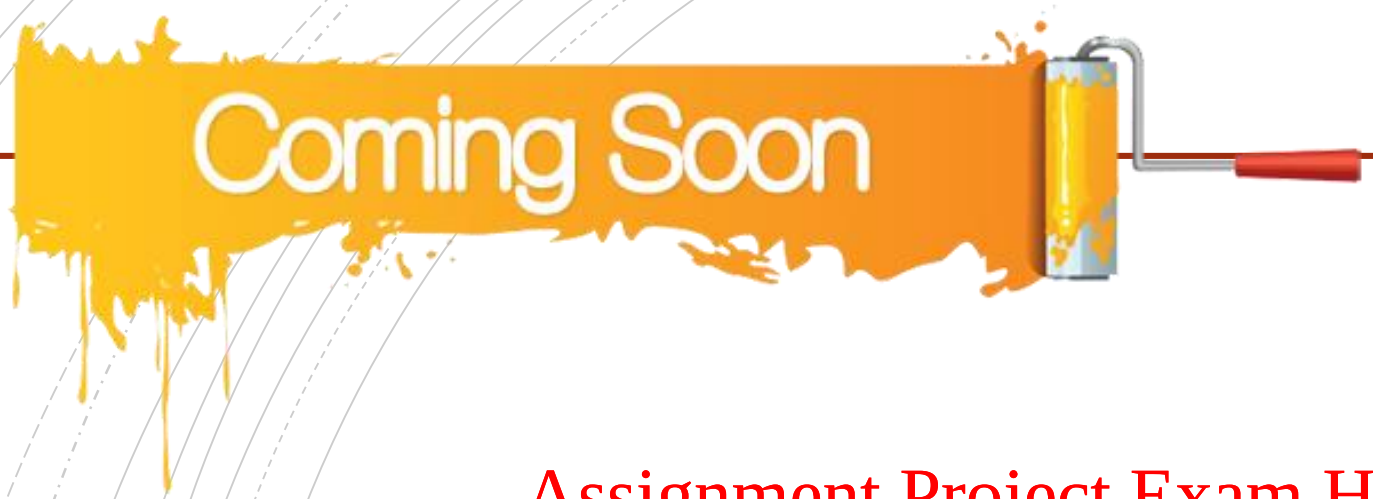
ORC – compareTo() TAKE 2

```
public class Orc implements Comparable<Orc> {
    private String name;
    private Integer height;
    private Weapon w;
    public int compareTo(Orc o
        int result = this.w.com
        if(result==0) {
            result = this.height.compareTo(o.h
        }
        if(result == 0) {
            result = this.name.compareTo(o.name);
        }
        return result;
    }
}
```


- We can also use compareTo() to compare multiple characteristics.
- Generally, it is better to reuse existing code than to write our own. Thus, in this case, we can use the *compareTo()* methods from other classes to.

TO RECAP

- `Comparable` defines a natural ordering.
- If you define a new d makes sense to you, then you should implement `Compar` ne a natural ordering on objects of such type.



In the next

- Iterable