Programming Technologies and Paradigms. Lab 2 presential activity. Groups 1 and 4.

Open `interfaces` within `inheritance.polymorphism` solution, carefully study how interfaces work in C#.
Using this interface:

```
namespace Intervals
{
    /// <summary>
    /// Returns the size: width for interval, area for bar,...
    /// </summary>
    public interface IMedible
    {
       double? Size();
    }
}
```

Implement these classes:
Interval

- Attributes: two nullable double (see `nullable` in `generics`) representing interval extremes, properties as needed. Empty interval attributes are both null.
- Default constructor: assigns null to both attributes.
- Constructor from two nullable double.
- Check, protected boolean method, checks the class invariant. Returns true if the invariant holds, false otherwise: left extreme must be less or equal than the right extreme unless both are null. It is not allowed an interval with only one attribute null.
- Size, returns a nullable double. For the empty interval, returns null. For non empty intervals returns the difference between the right and left sides. See `Compare` in project `interfaces`, solution `inheritance.polymorphism`.
- ToString.
- Equals and GetHashCode.
- One of theese: operator * or a method that returns the intersection of two intervals.

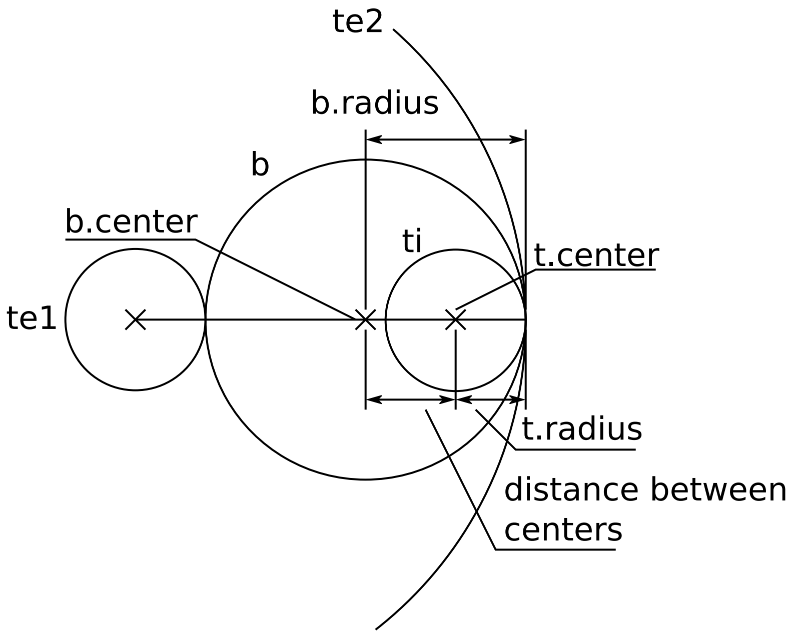
Bar: Inherits from Interval, represents a bar, as the ones found in a spreadsheet bar plot.

- Attribute: height, double, greater or equal to zero. Properties as needed.
- Default constructor, assigns 0 to height. Constructor from three double (two of them nullable, the Interval extremes).
- Private method Check, checks the class invariant. It is possible to implement it using Interval Check and `base`.
- ToString.
- Equals and GetHashCode.
- Size, returns bar area as a nullable double, it is possible to implement it calling `base Size`: height multiplied by base interval width.
- One of these: operator * or a method that returns the intersection of two bars, it is defined as the intersection of both bars interval attributes and the minimum height.

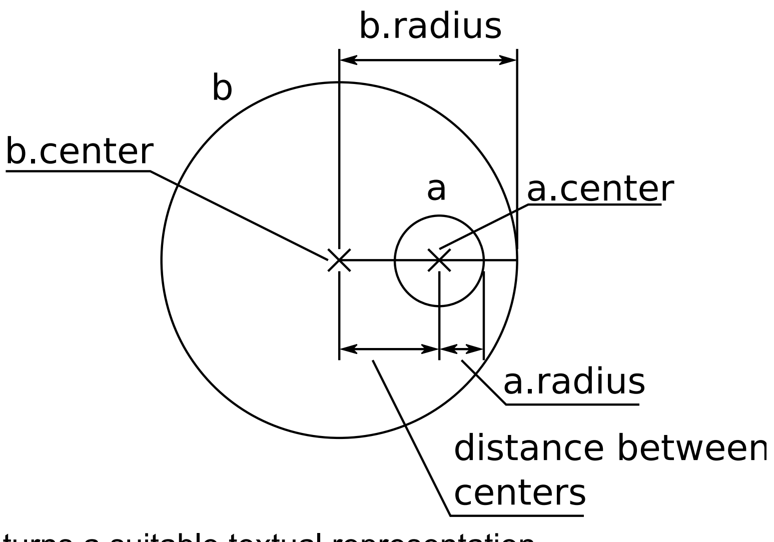
Pay attention to warnings if any: rethink modifiers for some methods.
Use assertions to verify invariants and postconditions. Use exceptions to treat and handle preconditions, see `exceptions` solution.
Write a console application with a polymorphic method HighestSize with two IMedible parameters, returning a reference to the biggest one (the one with the biggest Size value).
Check that all the implemented features (methods, operators,...) work as expected.

Optional: This is a composition exercise. Create a class library project, define Circumference class with the following attributes, methods and operators:

1. Attribute Center, Point2d type.
2. Attribute Radius, double type.
3. Method TangentWithCenter, takes a Point2d parameter, returns the circumference with given center tangent to the caller. If there are two possibilities choose any of them. The following drawing will help you with this.



4. Operator <, returns true if the left operand fits inside the left operand, false otherwise. After trying to build the project, complete with the requested operator. Again, the following drawing illustrates the procedure.



5. ToString, returns a suitable textual representation.
6. Constructors, destructor, properties, etc. and calls to implemented methods from a console application.

Upload the mandatory activity before the next Lab.