Separation of Responsibilities

Ridi Ferdiana | ridi@acm.org

Version 1.1.0

Overview

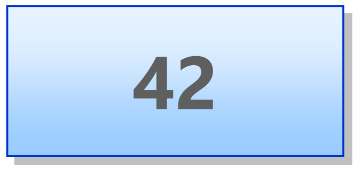
- Types in Object Oriented Design
- Conversion Model in Object Oriented
- Five Principles in Object Oriented Design

Types in Object Oriented Design

Value Types VS Reference Types

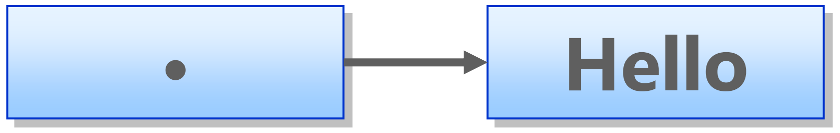
- Value types
  - The variable contains the value directly
  - Examples: **char**, **int**

```
int mol;
mol = 42;
```



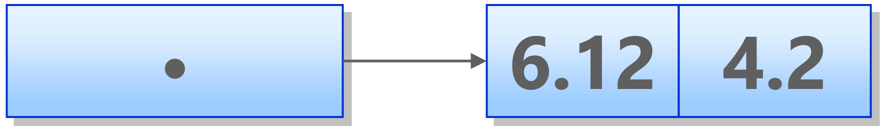
- Reference types
  - The variable contains a reference to the data
  - Data is stored in a separate memory area

```
string mol;
mol = "Hello";
```



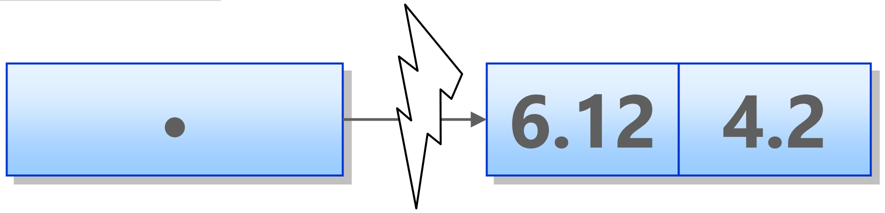
Declaring and Releasing Reference Variables

```
coordinate c1;
c1 = new coordinate();
c1.x = 6.12;
c1.y = 4.2;
```



- **Releasing reference variables**

```
c1 = null;
```

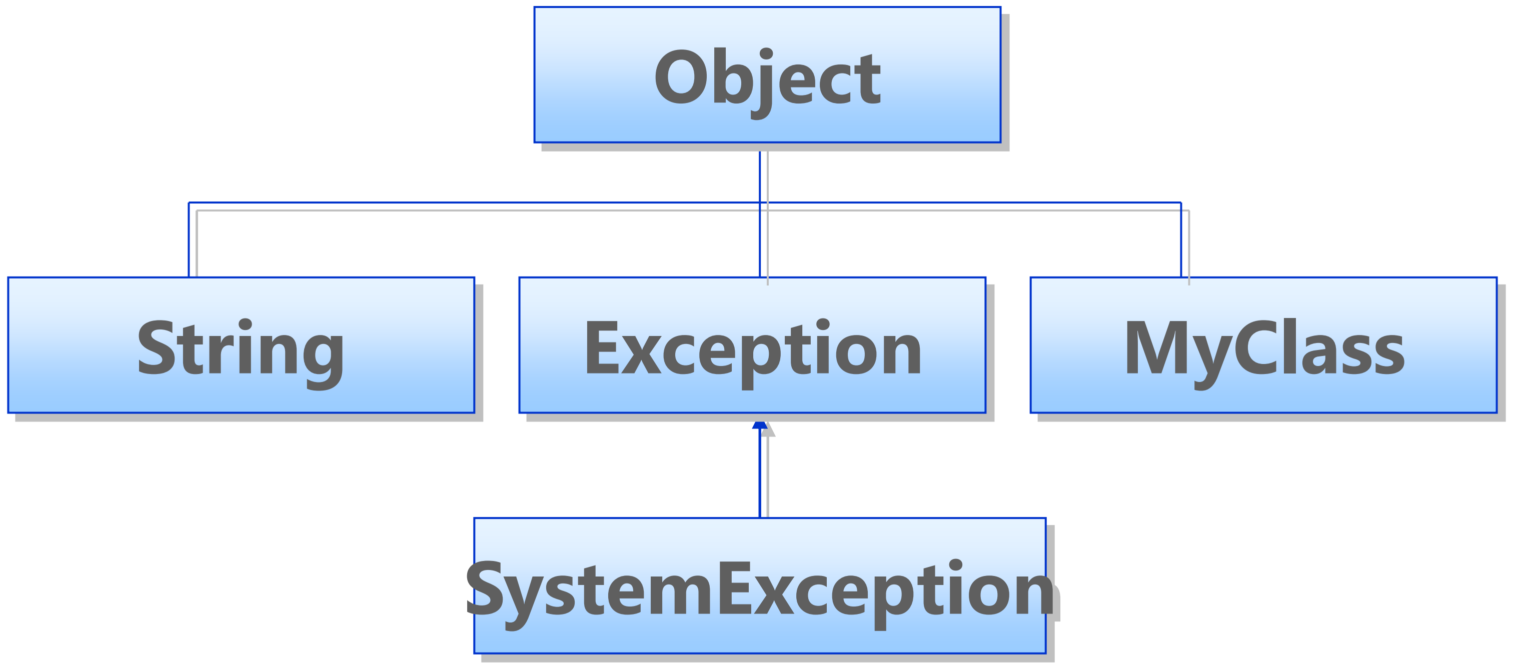


Conversion in Object Oriented

The object Type

Synonym for System.Object

Base class for all classes



Common Methods

Common methods for all reference types

- **ToString** method
- **Equals** method
- **GetType** method
- **Finalize** method

Converting Value Types

- Implicit conversions
- Explicit conversions
  - Cast operator
- Exceptions
- System.Convert class
  - Handles the conversions internally

The is Operator

Returns true if a conversion can be made

```
Bird b;
if (a is Bird)
    b = (Bird) a; // Safe
else
    Console.WriteLine("Not a Bird");
```

The as Operator

Converts between reference types, like cast

On error

- Returns null
- Does not raise an exception

```
Bird b = a as Bird; // Convert

if (b == null)
      Console.WriteLine("Not a bird");
```

Boxing and Unboxing

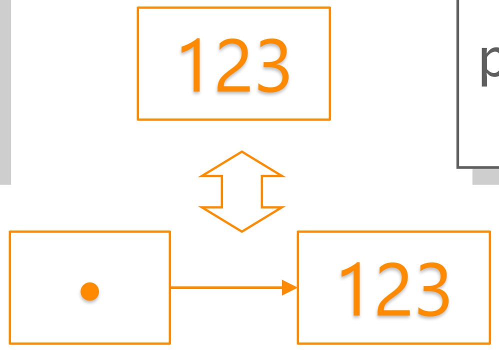
Unified type system

Boxing

Unboxing

Calling object methods on value types

```
int p = 123;
object box;
box = p;
```



```
p = (int)box;
```

Conversions and the object Type

The object type is the base for all classes

Any reference can be assigned to object

Any object variable can be assigned to any reference

- With appropriate type conversion and checks

The object type and is operator

```
object ox;
ox = a;
ox = (object) a;
ox = a as object;
```

```
b = (Bird) ox;
b = ox as Bird;
```

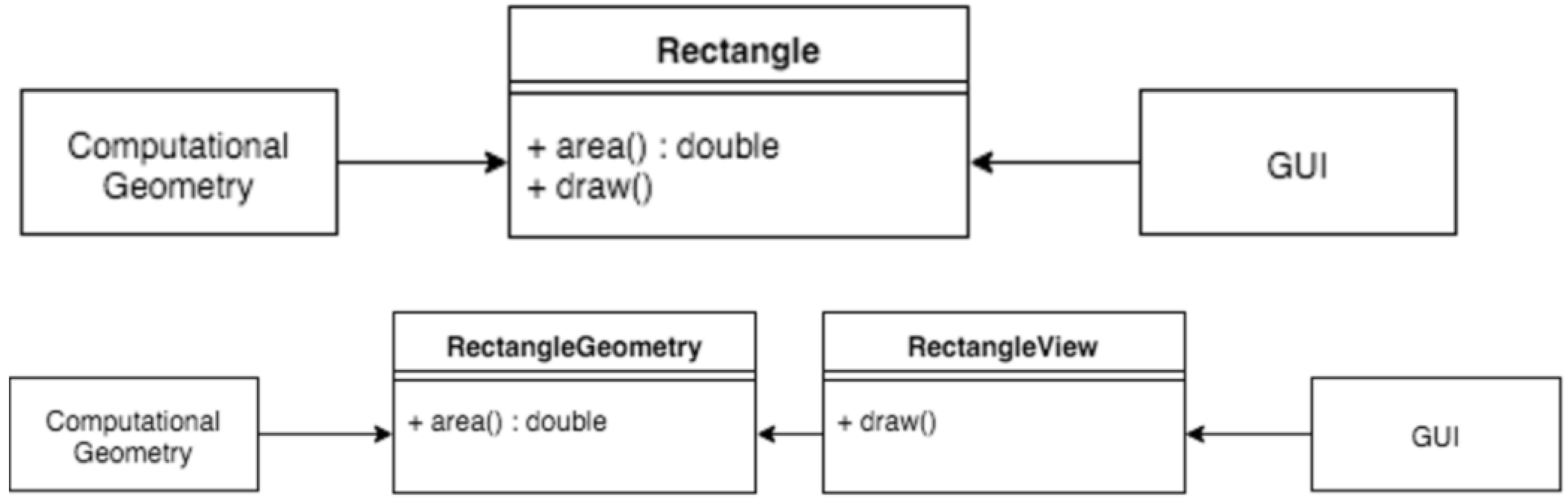
Five Principles in Object Oriented Design

SOLID Principles

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Single Responsibility Principle

- A class should have only one reason to change



Open Closed Principles

- the software entities (classes or methods) should be open for extension but closed for modification.
- Lowering the chance of producing bugs and domino effects

Demo

Open Closed Principles

Liskov Substitution Principle

- states that child class objects should be able to replace parent class objects without compromising application integrity
- Focuses in inheritance

Demo

Liskov Substitution Principle

Interface Segregation Principle

- The Interface Segregation Principle states that no client should be forced to depend on methods it does not use
- Focuses in interface

Demo

Interface Segregation Principle

Dependency Inversion Principle

- High-level modules should not depend on low-level modules, both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.

Demo

Dependency Inversion Principle

Review

- Types and References is the key performance of OO design
- Everything come from object, Everything can be converted and support the Object method
- Follow SOLID principles for better support of OO