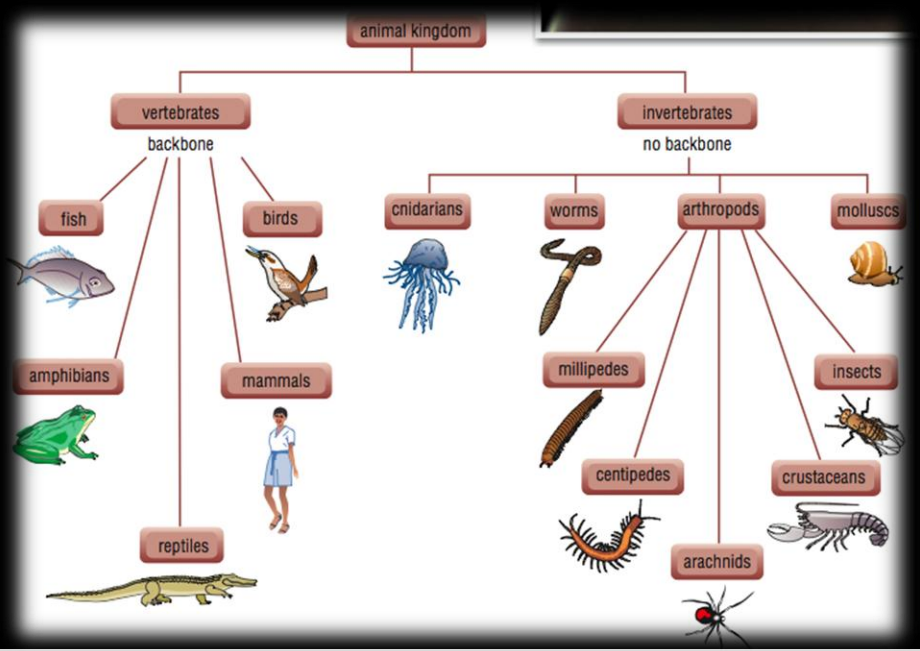


INHERITANCE & POLYMORPHISM

BME 121 2016

Rasoul Nasiri

Topics

- Class (fields, fields, methods) and UML
- Overloading methods
- Generics classes
- Inheritance
- polymorphism

Roles of Classes

- General class has some data + some methods
- UML class diagram

- Classes can be used to
  - Purely to model, store, and compute some data (Basic Class, e.g. RegistrationForm)
  - Model, store, and compute some data, as well as a relationship between one object and another (Relational Class)
  - Purely to store Helper methods & Constants (Helper Class, e.g. Math)
  - Model an advanced CS Data Structure (e.g. Linked List)

General Format:

ClassName
Fields
Methods

```
class Car
{
   private string make;
   private string model;
   private bool isOn; //on or off

   public Car( string make, string model )
   {
      this.make = make;
      this.model = model;
      wheels = new Wheel[4];
   }

   void start()
   {
      isOn = true;
   }
}
```

UML Class Diagram

- Show's a software's high level architecture
- Each class is a module or component of the software system

```
class Grumpy
{
  private string name;
  private string favQuote;

  public void SetName(string value)
  {
    this.name = value;
  }

  public string GetName()
  {
    return this.name;
  }
}
```

General Format:

ClassName
Fields
Methods

Notation details:
+ public
# protected
- Private
Static

Grumpy
- name : string - favQuote : string
+ SetName(value : string) : void + GetName() : string

Method overloading

- If we have a helper class for String operations
- Doing one operations in different ways
- Finding index of occurrence of a char
- We can have different versions of a method

StringOps
-
+ findIndex(a: string) : int + findIndex(a : string, last:bool) : int + findIndex(a: string, startingIndex:int) : int + findIndex(a: string, last: bool startingIndex:int) : int

Generics

- If we have a helper class for math operation.
- We can have different version of methods for different numerical types(int, double, long)

CompInt
- left: int - right: int
+ setLeft(a: int) : void + CheckEqual () : bool

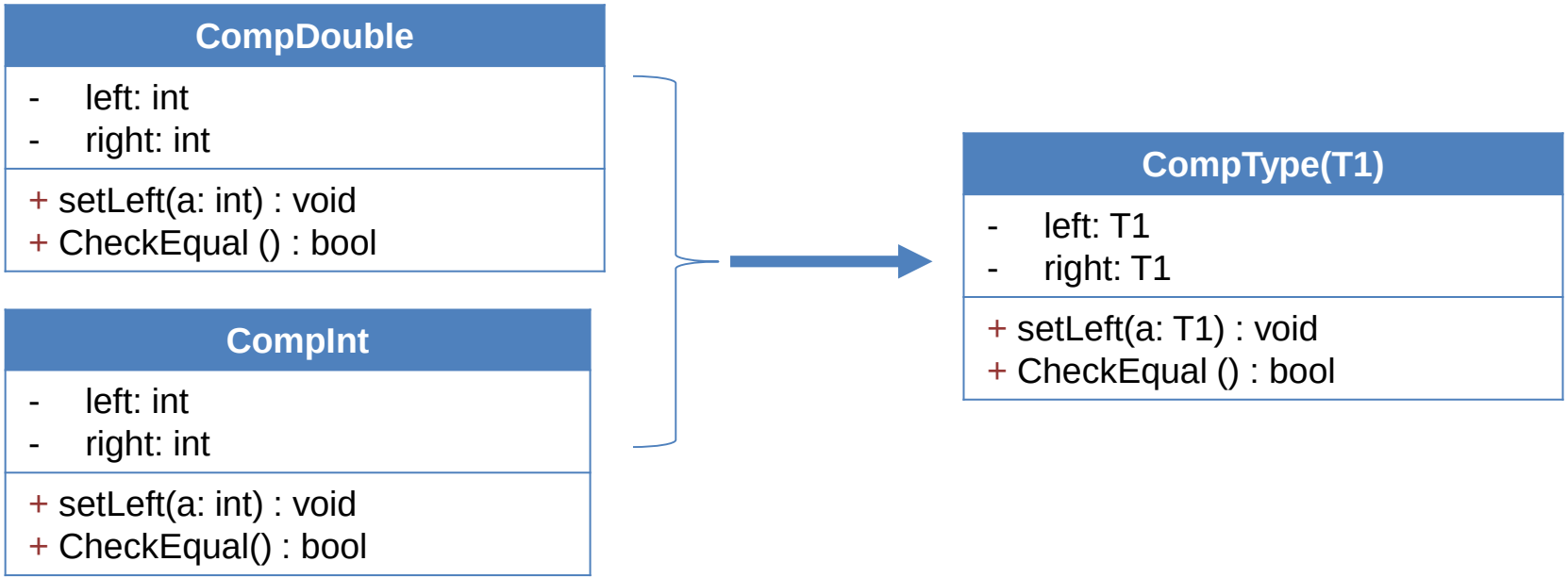
CompDouble
- left: int - right: int
+ setLeft(a: int) : void + CheckEqual () : bool

```
public class CompInt
{
  int left, right;
  void SetLeft (int a){
            left =    a;
  }
  double CheckEqual (){
            return left == right;
   }
}
```

```
public class CompDouble
{
   double left, right;
   double SetLeft (double a){
            left =    a;
  }
  double CheckEqual (){
            return left == right;
   }
}
```

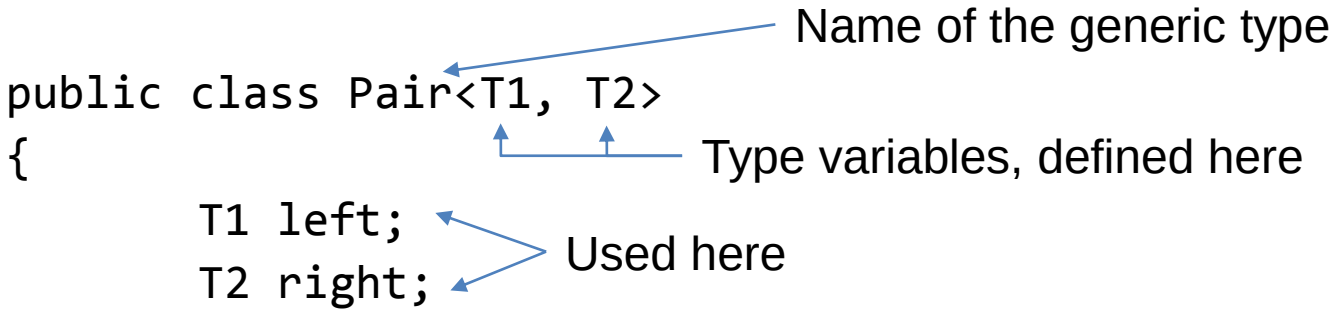
Generics

- We can merge these classes together.
- It is useful especially when there is a long list of methods.
- What is the short way to design such a class?



Generics

- A Generic Type is a kind of class that allows some customization to it's definition
- Commonly used to create flexible data structures



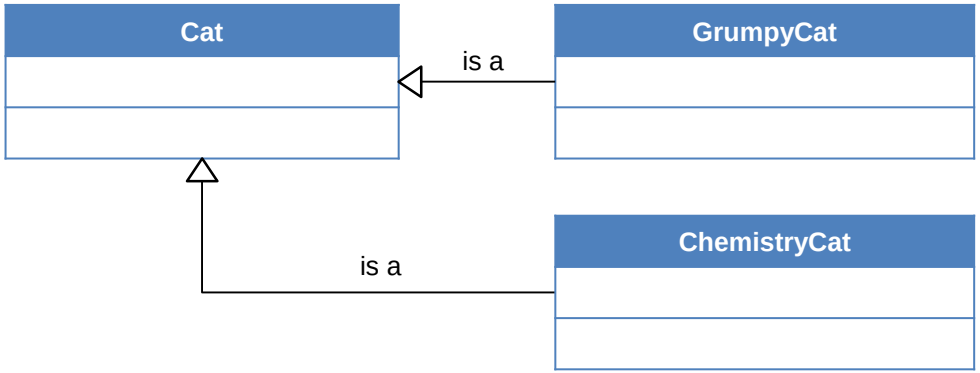
- This class can be used to create a relationship between a pair of variables.
  - Pair<string, int> firstPair = new Pair<string, int>("age", 20);
- When the program is compiled, these type variables are substituted for their actual values for each object of this class

Practice: Generic Comparison and Math

- Write a program to compare two values from int or double type and do some basic math on them.

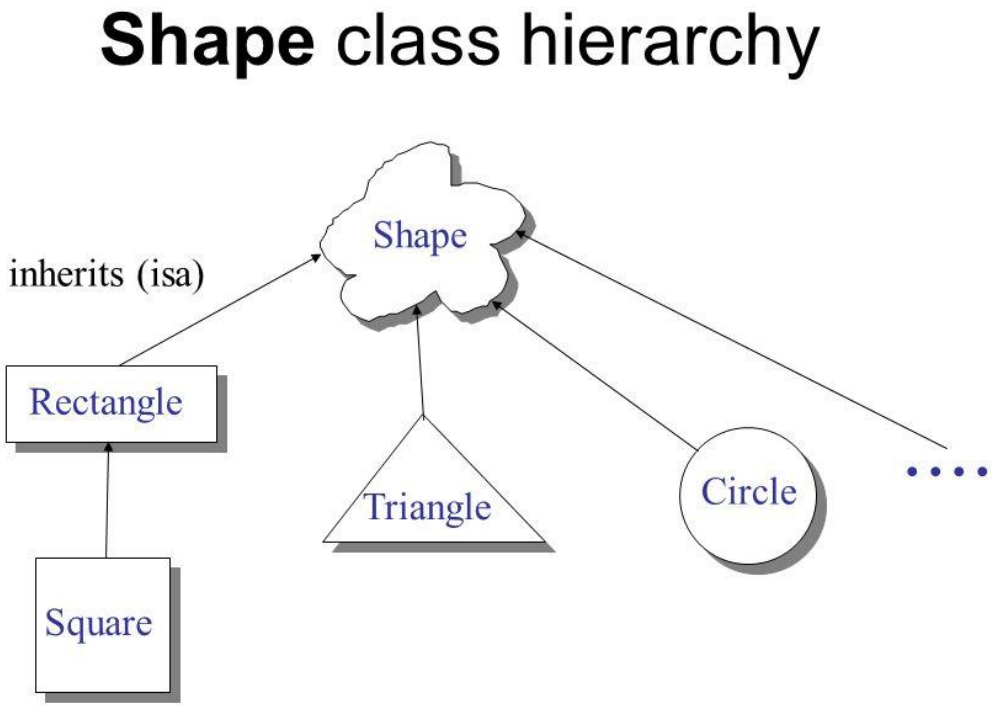
Inheritance

- White triangle points to the parent class

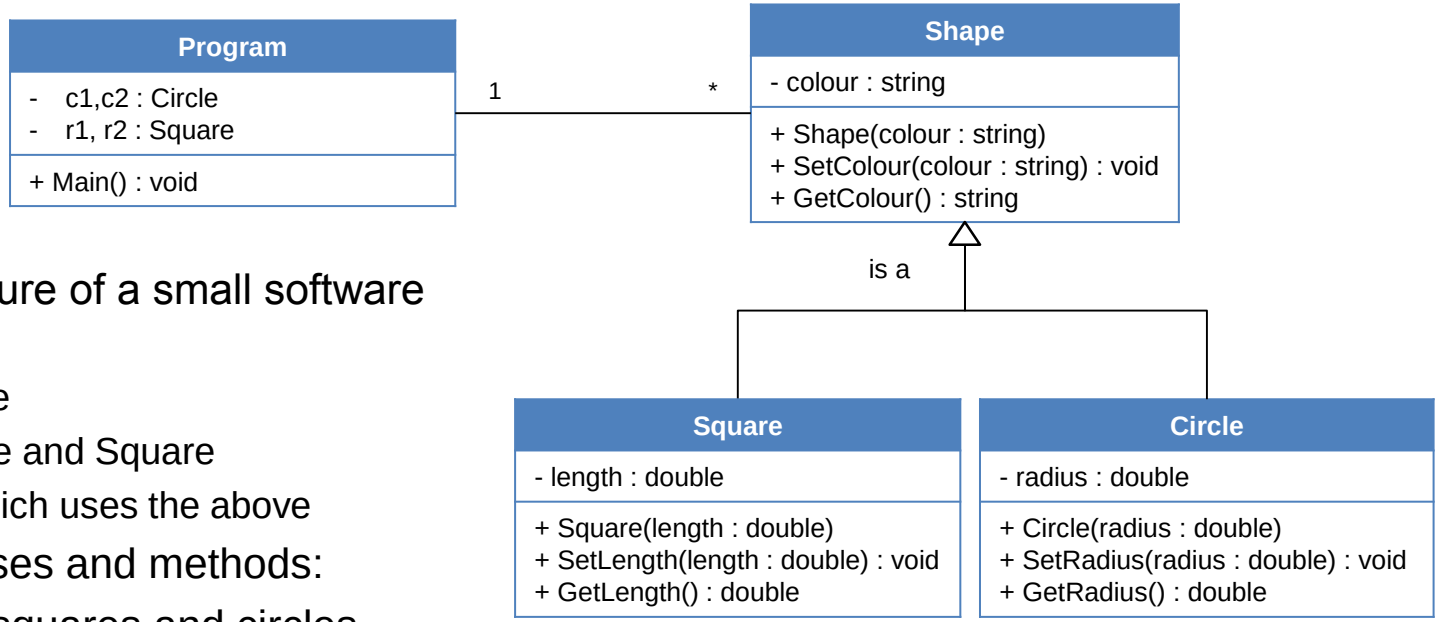



```
public class Cat
{
      public Cat() { }
}
public class GrumpyCat : Cat
{
      public GrumpyCat() { }
}
public class ChemistryCat : Cat
{
      public ChemistryCat() { }
}
```

Inheritance in shape concept



Inheritance Practice



- Here's the architecture of a small software which defines:
  - Parent class Shape
  - Child classes Circle and Square
  - Class Program, which uses the above
- Implement the classes and methods:
- In main, add a few squares and circles
- Get the colour of shapes and print them.

Polymorphic Method Invocation

- What you just witnessed, the ability to use the right version of a method depending on what class it belongs to, is called Polymorphic Method Invocation

Reference Configuration		
Reference Type	Object Type	Example
Parent Class	Parent Class	Cat c = new Cat();
Child Class	Child Class	Grumpy g = new Grumpy();
Parent Class	Child Class	Cat g = new Grumpy();
Child Class	Parent Class	Impossible