AlmaMate

Polymorphism

Objectives

- Understanding Polymorphism
- Types of Polymorphism

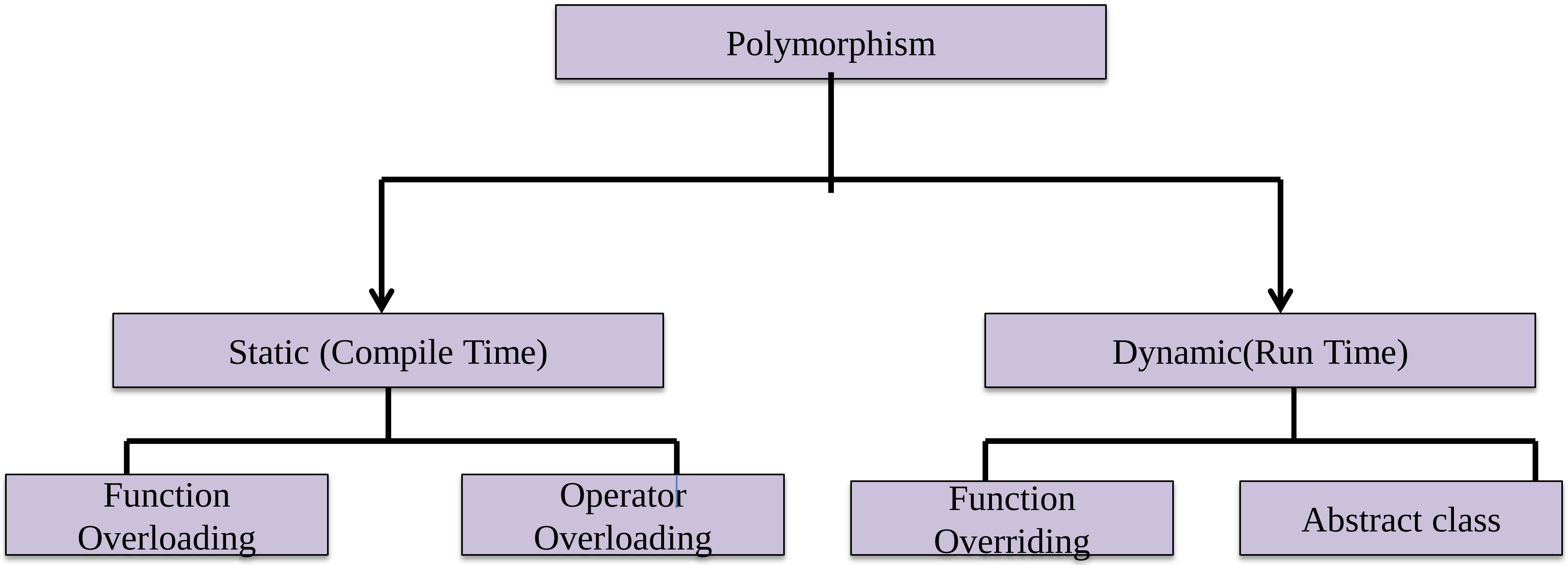
Defining Polymorphism

- Ability of different objects to respond to the same message in different ways is known as Polymorphism.
- Process of implementing base functionality of a parent object in a different way is known as Polymorphism.
- Ability of an object to behave differently depending on its type is Polymorphism.

Types of Polymorphism

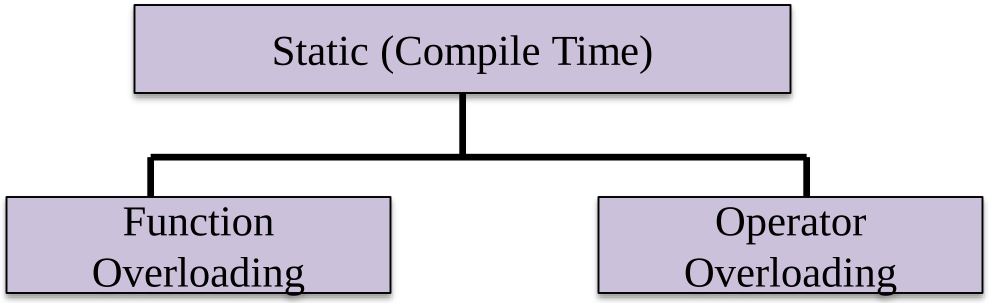
In C#, there are two types of Polymorphism, These are:

Static Polymorphism: Refers to an entity, which exists in various forms simultaneously.
Dynamic Polymorphism- Decision about function execution is made at run time.



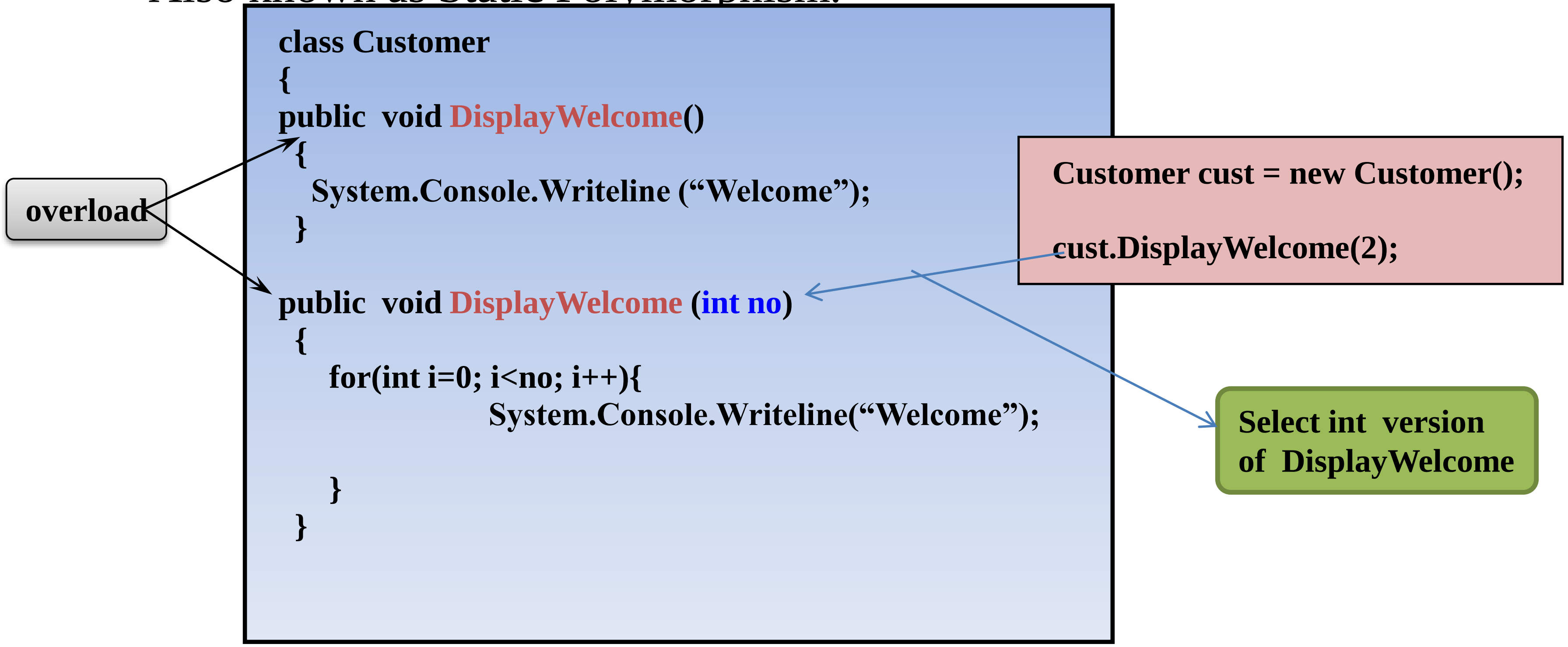
Static Polymorphism

- Is a process in which an entity, exists in various forms simultaneously.
- C# uses two approaches to implement Static Polymorphism. These are:
  - **Function overloading**: Allows creation of two or more functions having the same name. Each same name function must use different types , sequence, or number of parameters.
  - **Operator overloading**: Allows user-defined types such as structures and classes, to use overloaded operators for easy manipulation of their objects.



Function Overloading

- **Same name methods**
  - Parameters list must be different(Type, Number or Order)
  - Based on the passed parameters, compiler selects the appropriate method.
  - A method can be overloaded in the same class or in a subclass.
  - Also known as Static Polymorphism.



```
class Customer
{
public  void DisplayWelcome()
  {
    System.Console.Writeline ("Welcome");
  }

public  void DisplayWelcome (int no)
  {
      for(int i=0; i<no; i++){
                       System.Console.Writeline("Welcome");

      }
  }
```

```
Customer cust = new Customer();

cust.DisplayWelcome(2);
```

Constructor Overloading

```
Class Customer{
String name;
int custID ;
public Customer(int custID, string name){
            this.name = name;
            this.custID = custID
}
public Customer(int custID){
            this.custID = custID;
            name = “”;
}}
```

Class can have more than one constructor with constructor overloading

Based on arguments passed, compiler invokes the appropriate constructor.

Constructor invoking another constructor

```
class Calculator
{
int num1;
Int num2;
  public Calculator(int num1, int num2)
  {
    this.num1   = num1;
    this.num2 = num2;
  }
  public Calculator(int num1)
    :this(num1, 1)
  {
    //some code here
  }
}
```

call 2 argument constructor

Make the use of this( )
Put common code in the constructor that others call

Operator Overloading

- Operator overloading provides additional capabilities to C# operators when they are applied to user-defined data types.
- Only the predefined set of C# operators can be overloaded.

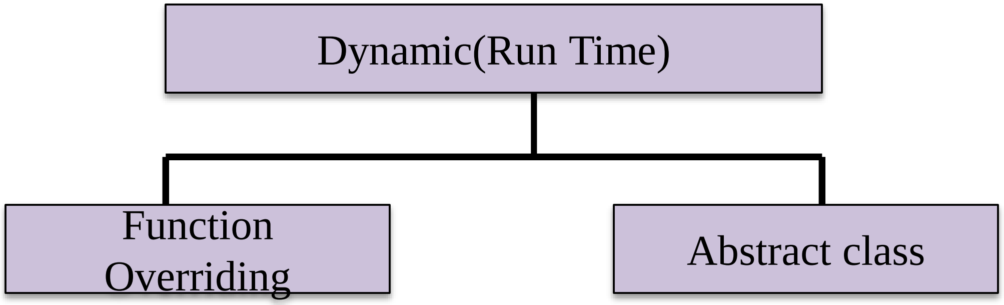
Need for Operator Overloading

- To use operators with user-defined data types, they need to be overloaded according to a programmer’s requirement.
- The following table describes the overload ability of the operators in C#.

Operators	**Description**
+, -, ! , ~, ++ , --	These unary operators take one operand and can be overloaded.
+, -, * , /, %	These binary operators take two operands and can be overloaded.
==, !=, <, >, <=, >=	The comparison operators can be overloaded.
&&, ||	The conditional logical operators cannot be overloaded directly, but they are evaluated using & and | which can be overloaded.
+=, -=, *=, /=, %=	The assignment operators cannot be overloaded.
=, ., ?:, ->, new, is, sizeof, typeof	These operators cannot be overloaded.

Dynamic Polymorphism

- It is the run time polymorphism in which, the decision about function execution is made at run time..
- C# uses two approaches to implement dynamic polymorphism:
  - **Function Overriding\Virtual function**:: Virtual function is a function which can be overridden in a derived class and the process of overriding a function in derived class is function overriding.
  - **Abstract class**: Act as a base class that consist of abstract members.



Function Overriding

- Is a process in which, a subclass redefine same name functions from the superclass
- By function overriding you can define/different behavior of a parent class method in   subclass.
- Call to overridden method is based on object type and will be decided at runtime

.

Function Overriding Example

```
 class Employee
       {
public virtual int CalculateSalary(int m ,int y)
       {
           int totalSalary = m * y;
           return totalSalary;
       }
}
class Manager : Employee
   {
       int bonus;
       public Manager(int bonus)
       {
           this.bonus = bonus;
       }
Contd..
```

```
public override int CalculateSalary(int m, int y)
      {
          int totalSalary = m * y + bonus;
          return totalSalary;
      }
      static void Main(string[] args)
      {
          Employee emp1 = new Employee();
          Employee emp2 = new Manager(2000);
System.Console.WriteLine(emp1.CalculateSalary(2, 10000));
System.Console.WriteLine(emp2.CalculateSalary(2, 10000));
      }
   }
```

It determined which method to run based on *object type* (Manager) instead of *reference type* (Manager)

Define *Employee* reference variable containing *Manager* object

Function Overriding key points

- An overridden method <u>must</u> have
  - the same name
  - the same number of parameters and types
  - the same return type as the overridden method
- For a method to be overridable without any compilation error/warning, it should be marked as virtual or abstract or override
- Methods declared as private, static, or sealed cannot be overridden
- Static method cannot override an instance method

AlmaMate

Thank You