ПЕРЕТВОРЕННЯ ТИПІВ ТА ПОЛІМОРФІЗМ ПІДТИПІВ

Питання 4.4.

Поліморфізм перетворення типів (Coercion polymorphism, casting)

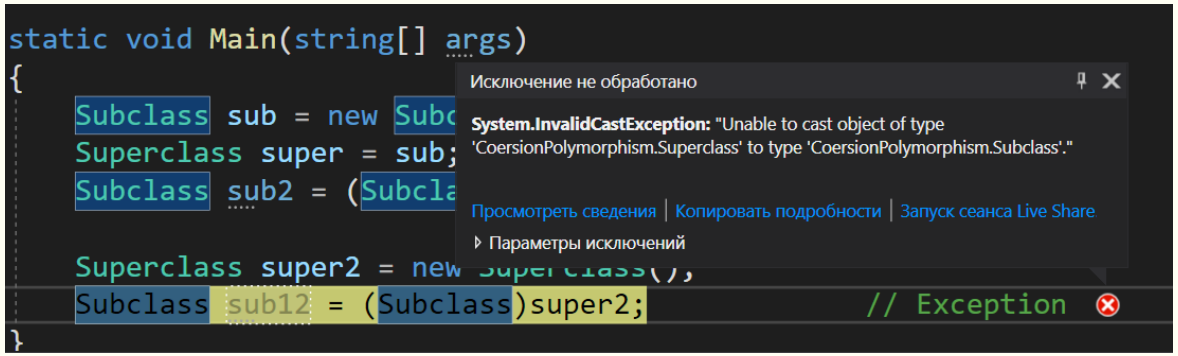
- Даний вид поліморфізму трапляється в випадку, коли об’єкт чи примітивне значення зводиться до іншого типу.
- ***Неявне перетворення типів (Implicit casting)*** відбувається, коли об’єкт присвоюється як значення змінної або параметр методу, що має сумісний тип.
  - Наприклад, типу, розташованого вище в ієрархії наслідування або типу інтерфейсу, що реалізує клас, з якого конструюється відповідний об’єкт.
  - У таких ситуаціях не потрібно додаткової роботи розробника – перетворення типів відбувається автоматично.
- ***Явне перетворення типів (зведення типів, Explicit casting)*** повинно застосовуватись, коли перетворення відбувається в зворотному напрямку відносно неявного перетворення типів.
  - Наприклад, при зведенні об’єкта до типу одного з підкласів або з типу інтерфейсу до типу класу.
  - Для виконання зведення типів використовується спеціальний оператор зведення.

Поліморфізм перетворення типів (Coercion polymorphism, casting)

```
class Superclass { }

class Subclass : Superclass { }

class Program
{
    static void Main(string[] args)
    {
        Subclass sub = new Subclass();
        Superclass super = sub;           // Implicit cast
        Subclass sub2 = (Subclass)super;  // Explicit cast


    }
}
```

- Зведення типів корисне, проте може бути проблемним, коли змінна, яка зводиться, несумісна з цільовим типом.
  - У такому випадку викидається виняток.

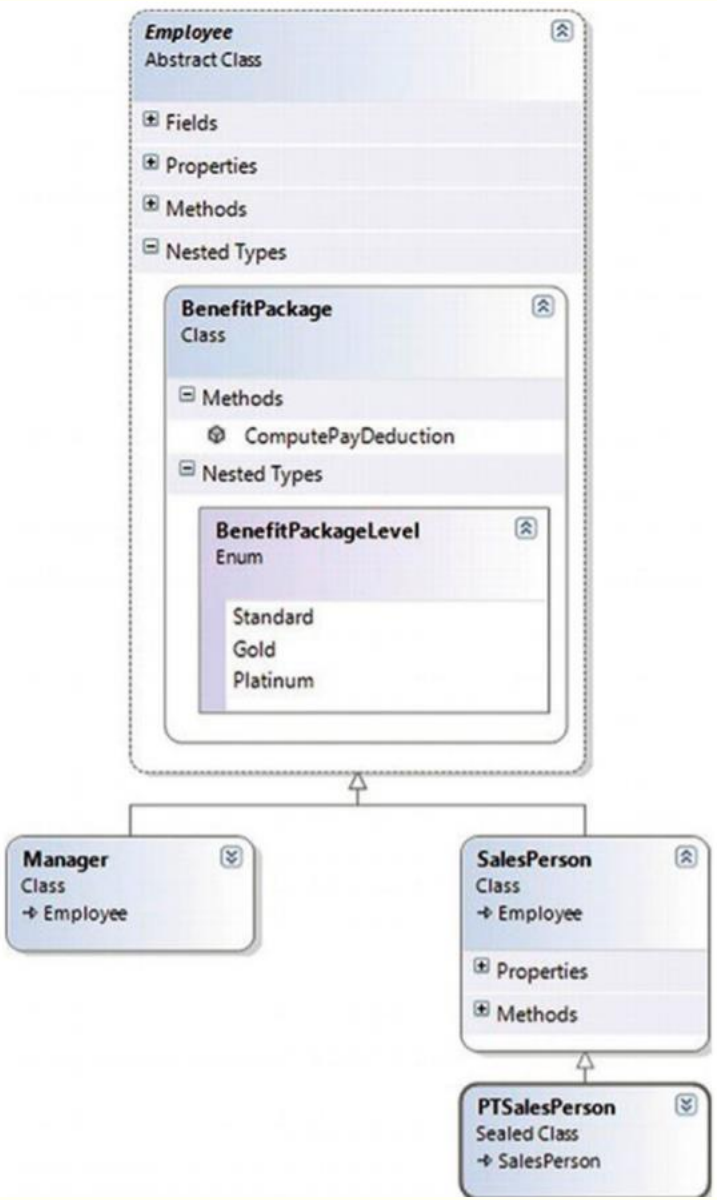
```
Superclass super2 = new Superclass();
Subclass sub12 = (Subclass)super2;   // Exception
```



Оператор as

```
Subclass sub3 = new Subclass();
Superclass super3 = sub3;
Subclass sub23 = super3 as Subclass;
if (sub == null)
{
    Console.WriteLine("Несумісні типи!");
}
```

- C# містить ще один спосіб виконання зведення типів: за допомогою оператора "as".
  - На відміну від явного перетворення, якщо зведення неможливе через несумісність типів, оператор не викидає виняток.
  - Замість цього результуюча змінна міститиме null.
- Хоч це хороший приклад захисного програмування, C# надає ключеве слово as для швидкого визначення сумісності одного типу з іншим під час виконання.



Правила зведення до базового та похідного класу

- Повернемось до ієрархії класів Employee
  - На платформі .NET початковим базовим класом є System.Object.
  - Тому все, що створюється, “являється” Object і може трактуватись як таке.
  - Таким чином, в об’єктній змінній можна зберігати посилання на екземпляр будь-якого типу:

```
static void CastingExamples()
{
  // Manager "является" System.Object, поэтому можно сохранять
  // ссылку на Manager в переменной типа object.
  object frank = new Manager("Frank Zappa", 9, 3000, 40000, "111-11-1111", 5);
}
```

- Типи Manager, SalesPerson и PTSalesPerson розширяють клас Employee, тому можна зберігати будь-який із цих об’єктів у допустимому посиланні на базовий клас.
  - Тому наступний код також коректний:

```
static void CastingExamples()
{
  // Manager "является" System.Object, поэтому можно сохранять
  // ссылку на Manager в переменной типа object.
  object frank = new Manager("Frank Zappa", 9, 3000, 40000, "111-11-1111", 5);

  // Manager также "является" Employee.
  Employee moonUnit = new Manager("MoonUnit Zappa", 2, 3001, 20000, "101-11-1321", 1);

  // PTSalesPerson "является" SalesPerson.
  SalesPerson jill = new PTSalesPerson("Jill", 834, 3002, 100000, "111-12-1119", 90);
}
```

Перше правило зведення між типами класів

- Коли два класи пов’язані відношенням “являється”, завжди можна безпечно зберегти породжений тип у посиланні базового класу.
  - Формально це називається неявним перетворенням типів, оскільки воно “просто працює” в відповідності з законами наслідування.
  - Це дозволяє будувати кілька потужних програмних конструкцій.
  - Наприклад, припустимо, що в поточному класі Program визначено новий метод:

```
static void GivePromotion(Employee emp)
{
  // Повысить зарплату...
  // Предоставить место на парковке компании...
  Console.WriteLine("{0} was promoted!", emp.Name);
}
```

Перше правило зведення між типами класів

- Поскольку этот метод принимает единственный параметр типа Employee, можно эффективно передавать этому методу любого наследника от класса Employee, учитывая отношение “является”:

```
static void CastingExamples()
{
  // Manager "является" System.Object, поэтому можно сохранять
  // ссылку на Manager в переменной типа object.
  object frank = new Manager("Frank Zappa", 9, 3000, 40000, "111-11-1111", 5);

  // Manager также "является" Employee.
  Employee moonUnit = new Manager("MoonUnit Zappa", 2, 3001, 20000, "101-11-1321", 1);
  GivePromotion(moonUnit);

  // PTSalesPerson "является" SalesPerson.
  SalesPerson jill = new PTSalesPerson("Jill", 834, 3002, 100000, "111-12-1119", 90);
  GivePromotion(jill);
}
```

- код компилируется благодаря неявному приведению от типа базового класса (Employee) к производному классу.
  - Однако что, если нужно также вызвать метод GivePromotion() для объекта frank (хранимого в данный момент в обобщенной ссылке System.Object)?

- Если вы передадите объект frank непосредственно в GivePromotion(), как показано ниже, то получите ошибку на этапе компиляции:

```
// Ошибка!
object frank = new Manager("Frank Zappa", 9, 3000, 40000, "111-11-1111", 5);
GivePromotion(frank);
```

  - Проблема в том, что предпринимается попытка передать переменную, которая является не Employee, а более общим объектом System.Object.
  - Поскольку в цепочке наследования он находится выше, чем Employee, компилятор не допустит неявного приведения, стараясь обеспечить максимально возможную безопасность типов.

Друге правило

- Несмотря на то что вы можете определить, что объектная ссылка указывает на совместимый с Employee класс в памяти, компилятор этого сделать не может, поскольку это не будет известно вплоть до времени выполнения.
  - Чтобы удовлетворить компилятор, понадобится выполнить явное приведение.
- Второе правило приведения гласит: необходимо явно выполнять приведение “вниз”, используя операцию приведения C#.
  - Базовый шаблон, которому нужно следовать при выполнении явного приведения, выглядит примерно так:
  - (класс_к_которому_нужно_привести) существующая_ссылка
  - Таким образом, чтобы передать переменную object методу GivePromotion (), потребуется написать следующий код:
  - // Правильно!
  - GivePromotion((Manager)frank);

Нові питання

- Учитывая, что метод GivePromotion () был спроектирован для приема любого возможного типа, производного от Employee, может возникнуть вопрос: как этот метод может определить, какой именно производный тип был ему передан?
- И, кстати, если входной параметр имеет тип Employee, как получить доступ к специализированным членам типов SalesPerson и Manager?

Ключевое слово is

```
static void GivePromotion(Employee emp)
{
  Console.WriteLine("{0} was promoted!", emp.Name);
  if (emp is SalesPerson)
  {
    Console.WriteLine("{0} made {1} sale(s)!", emp.Name,
      ((SalesPerson)emp).SalesNumber);
    Console.WriteLine();
  }
  if (emp is Manager)
  {
    Console.WriteLine("{0} had {1} stock options...", emp.Name,
      ((Manager)emp).StockOptions);
    Console.WriteLine();
  }
}
```

- В дополнение к ключевому слову as, в C# предлагается ключевое слово is, которое позволяет определить совместимость двух типов.
  - В отличие от ключевого слова as, если типы не совместимы, ключевое слово is возвращает false, а не ссылку null.

Поліморфізм включення (поліморфізм підтипів, subtyping, inclusion polymorphism)

- Поліморфізм підтрипів часто розглядається як частина наслідування.
  - Він надає підкласу спосіб визначення власної версії методу, описаного в його базовому класі, використовуючи процес ***заміщення (overriding) методу***.
  - Якщо базовий клас бажає визначити метод, який може бути (необов'язково) заміщеним у підкласі, він повинен відмітити його ключовим словом virtual.
  - Такі методи називаються ***віртуальними методами***.

```
partial class Employee
{
  // Этот метод теперь может быть переопределен производным классом.
  public virtual void GiveBonus(float amount)
  {
    Pay += amount;
  }
  ...
}
```

```
class SalesPerson : Employee
{
  ...
  // Бонус продавца зависит от количества продаж.
  public override void GiveBonus(float amount)
  {
    int salesBonus = 0;
    if (SalesNumber >= 0 && SalesNumber <= 100)
      salesBonus = 10;
    else
    {
      if (SalesNumber >= 101 && SalesNumber <= 200)
        salesBonus = 15;
      else
        salesBonus = 20;
    }
    base.GiveBonus(amount * salesBonus);
  }
}
```

```
class Manager : Employee
{
  ...
  public override void GiveBonus(float amount)
  {
    base.GiveBonus(amount);
    Random r = new Random();
    StockOptions += r.Next(500);
  }
}
```

Заміщення методу

- Когда класс желает изменить реализацию деталей виртуального метода, он делает это с помощью ключевого слова override.
  - Например, SalesPerson и Manager могли бы переопределить GiveBonus(), (предполагая, что PTSalesPerson не будет переопределять GiveBonus (), а потому просто наследует версию, определенную SalesPerson).
  - Обратите внимание на применение каждым переопределенным методом стандартного поведения через ключевое слово base.
  - Таким образом, полностью повторять реализацию логики GiveBonus() вовсе не обязательно, а вместо этого можно повторно использовать (и, возможно, расширять) стандартное поведение родительского класса.

Нехай поточний метод DisplayStatus() класу Employee оголошено віртуальним

- Кожний підклас може заміщати цей метод, розраховуючи на відображення кількості продажів (для продавців) і поточних опціонів на акції (для менеджерів). Наприклад,

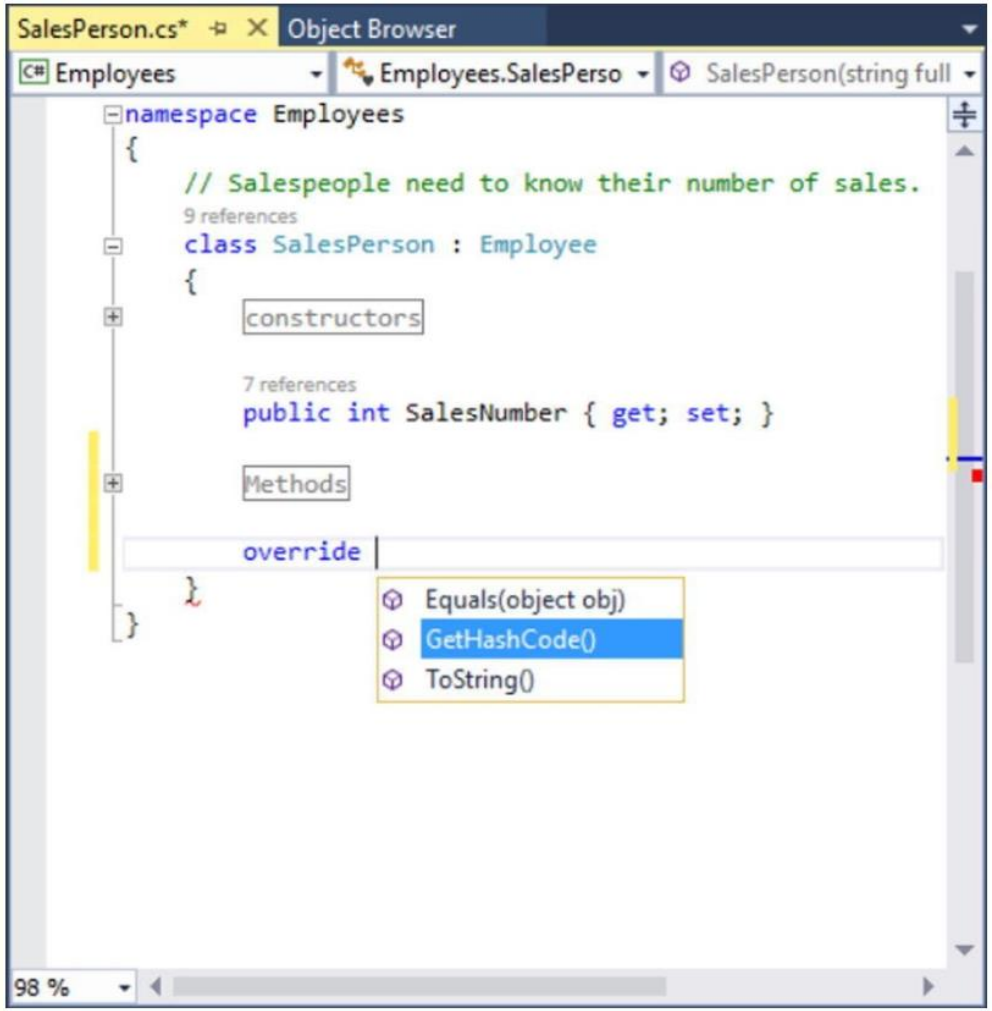
```
public override void DisplayStats()
{
  base.DisplayStats();
  Console.WriteLine("Number of Stock Options: {0}", numberOfOptions);
}
```

- Теперь кожний екземпляр об’єкта веде себе як більш незалежна сутність

```
static void Main(string[] args)
{
  Console.WriteLine("***** The Employee Class Hierarchy *****\n");
  // Лучшая система бонусов!
  Manager chucky = new Manager("Chucky", 50, 92, 100000, "333-23-2322", 9000);
  chucky.GiveBonus(300);
  chucky.DisplayStats();
  Console.WriteLine();
  SalesPerson fran = new SalesPerson("Fran", 43, 93, 3000, "932-32-3232", 31);
  fran.GiveBonus(200);
  fran.DisplayStats();
  Console.ReadLine();
}
```

```
***** The Employee Class Hierarchy *****

Name: Chucky
ID: 92
Age: 50
Pay: 100300
SSN: 333-23-2322
Number of Stock Options: 9337

Name: Fran
ID: 93
Age: 43
Pay: 5000
SSN: 932-32-3232
Number of Sales: 31
```

Заміщення віртуальних членів у Visual Studio



- При заміщенні члена класу необхідно пам’ятати типи всіх параметрів, а також угоди щодо передачі параметрів (ref, out і params).
  - У Visual Studio доступна корисна можливість: якщо набрати слово override всередині контексту типу класа (і натиснути пробіл), то IntelliSense автоматично відобразить список усіх членів батьківського класу, доступних для заміщення.

Запечатування віртуальних членів

- Иногда требуется не запечатывать класс целиком, а просто предотвратить переопределение некоторых виртуальных методов в производных типах.
  - Например, предположим, что продавцы с частичной занятостью не должны получать определенные бонусы. Чтобы предотвратить переопределение виртуального метода GiveBonus () в классе PTSalesPerson, можно запечатать этот метод в классе SalesPerson следующим образом:

```
// Класс SalesPerson запечатал метод GiveBonus()!
class SalesPerson : Employee
{
  ...
  public override sealed void GiveBonus(float amount)
  {
    ...
  }
}
```

- SalesPerson действительно переопределяет виртуальный метод GiveBonus(), определенный в классе Employee, однако он явно помечен как sealed.
  - Поэтому попытка переопределения этого метода в классе PTSalesPerson приведет к ошибке на этапе компиляции

```
sealed class PTSalesPerson : SalesPerson
{
  public PTSalesPerson(string fullName, int age, int empID,
                       float currPay, string ssn, int numbOfSales)
    : base (fullName, age, empID, currPay, ssn, numbOfSales)
  {
  }
  // Ошибка! Этот метод переопределять нельзя!
  public override void GiveBonus(float amount)
  {
  }
}
```

Головний батьківський клас System.Object

- Возможно, вы уже заметили в предыдущих разделах, что базовые классы всех иерархий (Car, Shape, Employee) никогда явно не указывали свои родительские классы:
  - // Кто является родительским классом Car?
  - class Car {…}
- Класс Object определяет набор общих членов для каждого типа внутри инфраструктуры.
  - Фактически при построении класса, который явно не указывает своего родителя, компилятор автоматически наследует его от Object.
- Если нужно очень четко прояснить свои намерения, можно определить класс, производный от Object:
  - // Явное наследование класса от System.Object.
  - class Car : object {...}

- В следующем формальном определении C# обратите внимание, что некоторые из этих членов определены как virtual, а это говорит о том, что данный член может быть переопределен в подклассе, в то время как другие помечены как static

```
public class Object
{
  // Виртуальные члены.
  public virtual bool Equals(object obj);
  protected virtual void Finalize();
  public virtual int GetHashCode();
  public virtual string ToString();
  // Невиртуальные члены уровня экземпляра.
  public Type GetType();
  protected object MemberwiseClone();

  // Статические члены.
  public static bool Equals(object objA, object objB);
  public static bool ReferenceEquals(object objA, object objB);
}
```

Основні методи System.Object

Метод экземпляра	Назначение
`Equals()`	По умолчанию этот метод возвращает `true`, только если сравниваемые элементы ссылаются в точности на один и тот же объект в памяти. Таким образом, `Equals()` используется для сравнения объектных ссылок, а не состояния объекта. Обычно этот метод переопределяется, чтобы возвращать `true`, только если сравниваемые объекты имеют одинаковые значения внутреннего состояния. Следует отметить, что в случае переопределения `Equals()` потребуется также переопределить метод `GetHashCode()`, потому что эти методы применяются внутренне типами `Hashtable` для извлечения подобъектов из контейнера.
`MemberwiseClone()`	Этот метод возвращает полную (почленную) копию текущего объекта и часто используется для клонирования объектов (см. главу 8)
`Finalize()`	На данный момент можно считать, что этот метод (будучи переопределенным) вызывается для освобождения любых размещенных ресурсов перед удалением объекта. Сборка мусора CLR более подробно рассматривается в главе 9
`GetHashCode()`	Этот метод возвращает значение `int`, идентифицирующее конкретный экземпляр объекта
`ToString()`	Этот метод возвращает строковое представление объекта, используя формат *<пространство_имен>.<имя_типа>* (т.н. *полностью заданное имя*). Этот метод часто переопределяется в подклассе для возврата строки, состоящей из пар "имя/значение", которая представляет внутреннее состояние объекта, вместо полностью заданного имени
`GetType()`	Этот метод возвращает объект `Type`, полностью описывающий объект, на который в данный момент производится ссылка. Коротко говоря, это метод идентификации типа во время выполнения (Runtime Type Identification — RTTI), доступный всем объектам (подробно обсуждается в главе 15)

Демонстрація роботи

```
class Person { }
```

```
class Program
{
  static void Main(string[] args)
  {
    Console.WriteLine("***** Fun with System.Object *****\n");
    Person p1 = new Person();

    // Использовать унаследованные члены System.Object.
    Console.WriteLine("ToString: {0}", p1.ToString());
    Console.WriteLine("Hash code: {0}", p1.GetHashCode());
    Console.WriteLine("Type: {0}", p1.GetType());

    // Создать другую ссылку на p1.
    Person p2 = p1;
    object o = p2;

    // Указывают ли ссылки на один и тот же объект в памяти?
    if (o.Equals(p1) && p2.Equals(o))
    {
      Console.WriteLine("Same instance!");  // Один и тот же экземпляр
    }
    Console.ReadLine();
  }
}
```

- Стандартна реалізація ToString() повертає повністю задану назву поточного типу (ObjectOverrides.Person).
  - Кожнй проект C# визначає “кореневий простір імен”, назва якого співпадає з назвою проекта.
  - Вивід метода Main():

```
***** Fun with System.Object *****

ToString: ObjectOverrides.Person
Hash code: 46104728
Type: ObjectOverrides.Person
Same instance!
```

  - Хоч готова поведінка System.Object часто задовольняє всі потреби, спеціальні типи заміщають деякі з цих успадкованих методів.

Заміщення System.Object.ToString()

```
// Помните: Person расширяет Object.
class Person
{
  public string FirstName { get; set; }
  public string LastName { get; set; }
  public int Age { get; set; }

  public Person(string fName, string lName, int personAge)
  {
    FirstName = fName;
    LastName = lName;
    Age = personAge;
  }
  public Person(){}
}
```

- рекомендованный подход состоит в разделении двоеточиями пар “имя/значение” и взятии всей строки в квадратные скобки

```
public override string ToString()
{
  string myState;
  myState = string.Format("[First Name: {0}; Last Name: {1}; Age: {2}]",
    FirstName, LastName, Age);
  return myState;
}
```

- всегда нужно помнить, что правильное переопределение ToString() должно также учитывать все данные, определенные выше в цепочке наследования.
- Когда вы переопределяете ToString () для класса, расширяющего специальный базовый класс, первое, что следует сделать — получить возвращаемое значение ToString () от родительского класса, используя ключевое слово base.
- Получив строковые данные родителя, можно добавить к ним специальную информацию производного класса.

Метод Equals()

- Стандартное поведение Equals () заключается в проверке того, указывают ли две переменных на один и тот же объект в памяти.
  - Здесь создается новая переменная Person по имени p1.
  - В этот момент новый объект Person помещается в управляемую кучу.
  - Переменная p2 также относится к типу Person.
  - Однако вы не создаете новый экземпляр, а вместо этого присваиваете этой переменной ссылку p1.
  - Таким образом, p1 и p2 указывают на один и тот же объект в памяти, как и переменная o (типа object).
  - Учитывая, что p1, p2 и o указывают на одно и то же местоположение в памяти, проверка эквивалентности дает положительный результат.
- переопределим поведение Object.Equals() для работы с семантикой на основе значений.
  - по умолчанию Equals () возвращает true, только если два сравниваемых объекта ссылаются на один и тот же экземпляр объекта в памяти.
  - В классе Person может быть полезно реализовать Equals () для возврата true, когда две сравниваемых переменных содержат одинаковые значения (т.е. фамилию, имя и возраст).

```
public override bool Equals(object obj)
{
  if (obj is Person && obj != null)
  {
    Person temp;
    temp = (Person)obj;
    if (temp.FirstName == this.FirstName
        && temp.LastName == this.LastName
        && temp.Age == this.Age)
    {
      return true;
    }
    else
    {
      return false;
    }
  }
  return false;
}
```

- Установив, что вызывающий код передал размещенный Person, один из подходов состоит в реализации Equals () для выполнения сравнения поле за полем данных входного объекта с соответствующими данными текущего объекта.
  - Здесь производится сравнение значения входного объекта с внутренними значениями текущего объекта (обратите внимание на применение ключевого слова this).
  - Если имя, фамилия и возраст, записанные в двух объектах, идентичны, значит, эти два объекта содержат одинаковые данные, потому возвращается true.
  - Любые другие возможные результаты приводят к возвращению false.

- Хотя этот подход действительно работает, представьте, насколько трудоемкой была бы реализация специального метода Equals () для нетривиальных типов, которые могут содержать десятки полей данных.
  - Распространенным сокращением является использование собственной реализации ToString().
  - Если у класса имеется правильная реализация ToString(), которая учитывает все поля данных вверх по цепочке наследования, можно просто сравнить строковые данные объектов:

```
public override bool Equals(object obj)
{
  // Больше нет необходимости приводить obj к типу Person,
  // поскольку у всех имеется метод ToString().
  return obj.ToString() == this.ToString();
}
```

  - Обратите внимание, что в этом случае нет необходимости проверять входной аргумент на принадлежность к корректному типу (в нашем примере — Person), поскольку все классы в .NET поддерживают метод ToString().

Заміщення System.Object.GetHashCode()

- Когда класс переопределяет метод Equals(), вы также обязаны переопределить стандартную реализацию GetHashCode() .
  - Говоря упрощенно, хеш-код — это числовое значение, представляющее объект как определенное состояние.
  - Например, если созданы две переменных string, хранящие значение Hello, они должны давать один и тот же хеш-код.
  - Однако если одна переменная string хранит строку в нижнем регистре (hello), должны быть получены разные хеш-коды.
- По умолчанию метод System. Obj ect. GetHashCode () для порождения хеш-значения использует текущее местоположение объекта в памяти.
  - Тем не менее, при построении специального типа, который нужно хранить в коллекции Hashtable (из пространства имен System.Collections), этот член должен быть всегда переопределен, поскольку Hashtable внутри вызывает Equals () и GetHashCode (), чтобы извлечь правильный объект.

- Хотя мы не собираемся помещать Person в System.Collections.Hashtable, для полноты давайте переопределим GetHashCode ().
  - Существует немало алгоритмов, которые могут применяться для создания хеш-кода, как весьма изощренные, так и не очень.
  - В большинстве случаев можно сгенерировать значение хеш-кода, полагаясь на реализацию System.String.GetHashCode().
  - Исходя из того, что класс String уже имеет хороший алгоритм хеширования, использующий символьные данные String для сравнения хеш-значений: если вы можете идентифицировать часть данных полей класса, которая должна быть уникальной для всех экземпляров (вроде номера карточки социального страхования), просто вызовите GetHashCode () на этой части полей данных.
- Поскольку в классе Person определено свойство SSN, можно написать следующий код:

```
// Предположим, что имеется свойство SSN.
class Person
{
  public string SSN {get; set;}
  // Вернуть хеш-код на основе уникальных строковых данных.
  public override int GetHashCode()
  {
    return this.ToString().GetHashCode();
  }
}
```

```
// Возвратить хеш-код на основе значения ToString()
public override int GetHashCode()
{
  return this.ToString().GetHashCode();
}
```

Тестування модифікованого класу Person

```
static void Main(string[] args)
{
  Console.WriteLine("***** Fun with System.Object *****\n");
  // ПРИМЕЧАНИЕ: эти объекты идентичны для проверки
  // методов Equals() и GetHashCode().
  Person p1 = new Person("Homer", "Simpson", 50);
  Person p2 = new Person("Homer", "Simpson", 50);
  // Получить строковые версии объектов.
  Console.WriteLine("p1.ToString() = {0}", p1.ToString());
  Console.WriteLine("p2.ToString() = {0}", p2.ToString());
  // Проверить переопределенный метод Equals().
  Console.WriteLine("p1 = p2?: {0}", p1.Equals(p2));
  // Проверить хеш-коды.
  Console.WriteLine("Same hash codes?: {0}", p1.GetHashCode() == p2.GetHashCode());
  Console.WriteLine();
  // Изменить возраст p2 и проверить снова.
  p2.Age = 45;
  Console.WriteLine("p1.ToString() = {0}", p1.ToString());
  Console.WriteLine("p2.ToString() = {0}", p2.ToString());
  Console.WriteLine("p1 = p2?: {0}", p1.Equals(p2));
  Console.WriteLine("Same hash codes?: {0}", p1.GetHashCode() == p2.GetHashCode());
  Console.ReadLine();
}
```

```
***** Fun with System.Object *****
p1.ToString() = [First Name: Homer; Last Name: Simpson; Age: 50]
p2.ToString() = [First Name: Homer; Last Name: Simpson; Age: 50]
p1 = p2?: True
Same hash codes?: True
p1.ToString() = [First Name: Homer; Last Name: Simpson; Age: 50]
p2.ToString() = [First Name: Homer; Last Name: Simpson; Age: 45]
p1 = p2?: False
Same hash codes?: False
```

Статичні члени System.Object

- В дополнение к только что рассмотренным членам уровня экземпляра, в System.Object определены два очень полезных статических члена, которые также проверяют эквивалентность на основе значений или на основе ссылок.

```
static void StaticMembersOfObject()
{
  // Статические члены System.Object.
  Person p3 = new Person("Sally", "Jones", 4);
  Person p4 = new Person("Sally", "Jones", 4);
  Console.WriteLine("P3 and P4 have same state: {0}", object.Equals(p3, p4));
  Console.WriteLine("P3 and P4 are pointing to same object: {0}",
    object.ReferenceEquals(p3, p4));
}
```

  - Здесь можно просто передать два объекта (любого типа) и позволить классу System.Object автоматически определить детали.

ДЯКУЮ ЗА УВАГУ!

Наступне питання: