Наследование и переопределение методов (часть 1)

Оглавление

Одним из фундаментальных механизмов, лежащих в основе любого объектно-ориентированного языка, в том числе Java, является наследование. Наследование позволяет одним объектам получать характеристики других объектов. Представим себе ситуацию, когда на основе уже существующего, проверенного и работающего кода нужно создать новую программу. Есть два пути решения этой задачи. Во-первых, можно скопировать уже существующий код в новый проект внести необходимые изменения. Во-вторых, в новом проекте можно сделать ссылку на уже существующий код. Второй вариант во многих отношениях предпочтительнее, поскольку позволяет сэкономить время и усилия на создание нового кода и обеспечивает более высокую степень совместимости программ. Ведь если окажется, что базовый код необходимо доработать, то это достаточно сделать единожды: поскольку код инкапсулируется через ссылку, внесенные изменения вступят в силу автоматически во всех местах, где этот код используется.

Именно по этому принципу реализован механизм наследования.

С практической точки зрения наследование позволяет одним объектам получать (наследовать) свойства других объектов. Реализуется наследование путем создания классов на основе уже существующих классов. При этом члены класса, на основе которого создается новый класс, с некоторыми оговорками, автоматически включаются в новый класс. Кроме того, в создаваемый класс можно добавлять новые члены. Согласно общепринятой терминологии, класс, на основе которого создается новый класс, называется суперклассом. Новый создаваемый на основе суперкласса класс называется подклассом.

Создание подкласса

Создание подкласса практически не отличается от создания обычного класса, только при создании подкласса необходимо указать суперкласс, на основе которого создается подкласс.

Для реализации наследования в описании подкласса после имени класса указывается ключевое слово extends и имя суперкласса. Во всем остальном описание подкласса не отличается от описания обычного класса (то есть

класса, который создается «с нуля»). Синтаксис описания подкласса имеет вид:

```
class A extends B{
// код
}
```

В данном случае подкласс A создается на основе суперкласса B. В результате подкласс A получает (наследует) открытые и защищенные члены класса B.

Обращаем внимание, что в языке Java, в отличие от языка C++, отсутствует множественное наследование, то есть подкласс в Java может создаваться на основе только одного суперкласса. При этом в Java, как и в C++, существует многоуровневое наследование: подкласс, в свою очередь, может быть суперклассом для другого класса. Благодаря многоуровневому наследованию можно создавать целые цепочки связанных механизмом наследования классов. В листинге приведен пример создания подкласса:

```
//Создание подкласса
class A{ // Суперкласс
int i,j;
void showij(){
System.out.println("Поля i и j: "+i+" и "+j);}
}
class B extends A{ // Подкласс
int k;
void showk(){
System.out.println("Поле k: "+k);}
void sum(){
// Обращение к наследуемым полям:
System.out.println("Сумма i+j+k="+(i+j+k));}
}
class AB{
public static void main(String arg[]){
// Объект суперкласса:
A SuperObj=new A();
// Объект подкласса:
B SubObj=new B();
SuperObj.i=10;
SuperObj.j=20;
SuperObj.showij();
SubObj.i=7;
SubObj.j=8;
SubObj.k=9;
SubObj.showij();
SubObj.showk();
SubObj.sum();}
}
```

В программе описан суперкласс A, в котором объявлены два целочисленных поля i и j, а также метод showij() для отображения значений этих полей. На основе класса A создается класс B (подкласс суперкласса A). Непосредственно в теле класса B описано целочисленное поле k, а также методы showk()

и sum() для вывода значения поля k и вычисления суммы полей i, j и k. Обращаем внимание, что хотя поля i и j непосредственно в классе B не описаны, в классе B о них известно, поскольку они наследуются этим классом (то есть у класса B имеются целочисленные поля i и j) и к ним можно обращаться.

В методе main() класса AB создаются два объекта: объект SuperObj суперкласса A и объект SubObj подкласса B. Полям i и j объекта SuperObj присваиваются значения 10 и 20 соответственно, после чего с помощью метода showij() значения полей выводятся на экран.

Полям i, j и k объекта SubObj присваиваются целочисленные значения 7, 8 и 9.

Методом showij() отображаются значения полей i и j, а значение поля k отображается с помощью метода showk(). Наконец, сумма полей вычисляется методом sum(). Результат выполнения программы следующий:

```
Поля i и j: 10 и 20
Поля i и j: 7 и 8
Поле k: 9
Сумма i+j+k=24
```

Другими словами, ситуация такая, как если бы поля i и j, а также метод showij() были описаны в классе B. Достигается такой эффект благодаря наследованию.

Доступ к элементам суперкласса

Не все члены суперкласса наследуются в подклассе. Наследование не распространяется на закрытые члены суперкласса. Другими словами, в подклассе закрытые члены суперкласса недоступны. Напомним, что закрытые члены класса объявляются с ключевым словом private, а по умолчанию, если никакое ключевое слово не указано, члены класса считаются открытыми. Именно поэтому, несмотря на отсутствие ключевых слов, описывающих уровень доступа, в рассмотренном примере никаких проблем с наследованием не возникало.

Для иллюстрации того, что происходит при наследовании, когда суперкласс содержит закрытые члены, рассмотрим пример в листинге:

```
class MySuperClass{ // Суперкласс
    // Закрытое поле:
    private int a;
    // Закрытый метод:
    private void showa(){
    System.out.println("Поле a: "+a);}
    // Открытый метод:
    void seta(int n){
    a=n;
    showa();}
    }
    class MySubClass extends MySuperClass{ // Подкласс
    int b;
    void setall(int i,int j){
    seta(i);
    b=j;
    System.out.println("Поле b: "+b);}
    }
    class PrivateSuperDemo{
```

```
public static void main(String arg[]){
// Объект подкласса:
MySubClass obj=new MySubClass();
obj.setall(1,5);}
}
```

В результате выполнения этой программы получаем сообщения:

Поле a: 1

Поле b: 5

Рассмотрим подробнее программный код и особенности его выполнения. В первую очередь имеет смысл обратить внимание на суперкласс MySuperClass, в котором описывается закрытое (с идентификатором доступа private) целочисленное поле a и два метода. Закрытый метод showa() предназначен для отображения значения поля a. Открытый метод seta() позволяет присвоить значение закрытому полю a и вывести значение этого поля на экран — для этого в методе seta() вызывается метод showa(). Следовательно, при вызове открытого метода seta() выполняется обращение к закрытому полю a, причем как напрямую, так и через вызов закрытого метода showa().

В подклассе MySubClass описывается открытое целочисленное поле b и открытый метод setall(). Кроме того, классом MySubClass из класса MySuperClass наследуется открытый метод seta(). Закрытое поле a и закрытый метод showa() классом MySubClass не наследуются.

Объявленный непосредственно в классе MySubClass метод setall() вызывает, кроме прочего, наследуемый из класса MySuperClass метод seta(), который, в свою очередь, обращается к ненаследуемому полю a и ненаследуемому методу showa(). Может сложиться впечатление, что такой код некорректен, поскольку, например, при вызове метода setall() из объекта obj класса MySubClass делается попытка присвоить и считать значение для поля a, которого в объекте obj в принципе нет. Тем не менее код работает.

Все становится на свои места, если уточнить понятия «наследуется» и «не наследуется». Дело в том, что наследование членов суперкласса подразумевает, что эти поля доступны в подклассе. Другими словами, подкласс «знает» о существовании наследуемых членов, и к этим членам можно обращаться так, как если бы они были описаны в самом классе. Если же член классом не наследуется, то о таком члене класс ничего «не знает», и, соответственно, попытка обратиться к такому «неизвестному» для класса члену напрямую ведет к ошибке. Однако технически ненаследуемые члены в классе существуют, о чем свидетельствует хотя бы приведенный пример. Причина кроется в способе создания объектов подкласса. Дело в том, что при создании объекта подкласса сначала вызывается конструктор суперкласса, а затем непосредственно конструктор подкласса. Конструктором суперкласса выделяется в памяти место для всех членов объекта, в том числе и ненаследуемых.

Конструкторы и наследование

Если суперкласс и подкласс используют конструкторы по умолчанию (то есть ни в суперклассе, ни в подклассе конструкторы не описаны), то процесс создания объекта подкласса для программиста проходит обыденно — так

же, как создание объекта обычного класса. Ситуация несколько меняется, если конструктору суперкласса необходимо передавать аргументы. Возникает проблема: поскольку при создании объекта подкласса сначала автоматически вызывается конструктор суперкласса, в этот конструктор как-то нужно передать аргументы, даже если непосредственно конструктор подкласса может без них обойтись.

Все это накладывает некоторые ограничения на способ описания конструктора подкласса. Формально эти ограничения сводятся к тому, что в конструкторе подкласса необходимо предусмотреть передачу аргументов конструктору суперкласса (разумеется, если такая передача аргументов вообще требуется).

Технически решение проблемы сводится к тому, что в программный код конструктора подкласса добавляется инструкция вызова конструктора суперкласса с указанием аргументов, которые ему передаются. Для этого используется ключевое слово super, после которого в круглых скобках указываются аргументы, передаваемые конструктору суперкласса. Инструкция вызова конструктора суперкласса указывается первой командой в теле конструктора подкласса. Таким образом, общий синтаксис объявления конструктора подкласса имеет следующий вид:

```
конструктор_подкласса(аргументы1){
super(аргументы2); // аргументы конструктора суперкласса
// тело конструктора подкласса
}
```

Если в теле конструктора подкласса инструкцию super не указать вовсе, в качестве конструктора суперкласса вызывается конструктор по умолчанию (конструктор без аргументов). Пример описания конструкторов при наследовании приведен в листинге:

```
// Суперкласс:
class MySuperClass{
int a;
void showa(){
System.out.println("Объект с полем a="+a);}
// Конструкторы суперкласса:
MySuperClass(){
a=0;
showa();}
MySuperClass(int i){
a=i;
showa();}
}
// Подкласс:
class MySubClass extends MySuperClass{
double x;
void showx(){
System.out.println("Объект с полем x="+x);}
// Конструкторы подкласса:
MySubClass(){
super(); // Вызов конструктора суперкласса
x=0;
showx();}
MySubClass(int i,double z){
super(i); // Вызов конструктора суперкласса
x=z;
showx();}
```

```
}
class SuperConstrDemo{
public static void main(String[] args){
System.out.println("Первый объект:");
MySubClass obj1=new MySubClass();
System.out.println("Второй объект:");
MySubClass obj2=new MySubClass(5,3.2);}
}
```

В результате выполнения этой программы получаем последовательность сообщений:

Первый объект:
Объект с полем a=0
Объект с полем x=0.0
Второй объект:
Объект с полем a=5
Объект с полем x=3.2

Программа состоит из трех классов. В первом классе MySuperClass описано целочисленное поле a, метод showa() для отображения значения этого поля, а также два варианта конструкторов: без аргументов и с одним аргументом. В конструкторе без аргументов полю a присваивается нулевое значение. В конструкторе с аргументом полю присваивается значение аргумента. В обоих случаях с помощью метода showa() значение поля a выводится на экран.

На основе класса MySuperClass создается подкласс MySubClass. Непосредственно в классе описывается поле x типа double и метод showx() для отображения значения этого поля.

В подклассе определяются два конструктора: без аргументов и с двумя аргументами. В каждом из этих конструкторов с помощью инструкции super вызывается конструктор суперкласса. В конструкторе подкласса без аргументов командой super() вызывается конструктор суперкласса без аргументов. Если при создании объекта подкласса конструктору передаются два аргумента (типа int и типа double), то аргумент типа int передается аргументом конструктору суперкласса (командой super(i) в теле конструктора подкласса с двумя аргументами).

В главном методе программы создаются два объекта подкласса MySubClass. В первом случае вызывается конструктор без аргументов, во втором — конструктор с двумя аргументами.

Ссылка на элемент суперкласса

При наследовании могут складываться достаточно неоднозначные ситуации. Один из примеров такой ситуации — совпадение названия, наследуемого подклассом поля с названием поля, описанного непосредственно в подклассе. С формальной точки зрения подобная ситуация выглядит так, как если бы у подкласса было два поля с одним и тем же именем: одно поле собственно подкласса и одно, полученное «по наследству». Технически так оно и есть. В этом случае естественным образом возникает вопрос о способе обращения к

таким полям. По умолчанию если обращение выполняется в обычном формате, через указание имени поля, то используется то из двух полей, которое описано непосредственно в подклассе.

Рассмотрим пример, представленный в листинге:

```
// Суперкласс:
class MyClassA{
// Поле:
int number;
// Конструктор суперкласса:
MyClassA(){
number=0;
System.out.println("Создан объект суперкласса с полем "+number);}
// Отображение значения поля:
void showA(){
System.out.println("Поле number: "+number);}
}
// Подкласс:
class MyClassB extends MyClassA{
// Поле с тем же именем:
int number;
// Конструктор подкласса:
MyClassB(){
super(); // Вызов конструктора суперкласса
number=100;
System.out.println("Создан объект подкласса с полем "+number);}
// Отображение значения поля:
void showB(){
System.out.println("Поле number: "+number);}
}
class TwoFieldsDemo{
public static void main(String[] args){
// Создание объекта подкласса:
MyClassB obj=new MyClassB();
// Изменение значения поля:
obj.number=50;
// Отображение значения поля:
obj.showA();
obj.showB();
}}
```

Результат выполнения программы имеет вид:

Создан объект суперкласса с полем 0
Создан объект подкласса с полем 100
Поле number: 0
Поле number: 50

В классе MyClassA объявлены числовое поле number, метод showA() для отображения значения этого поля и конструктор без аргументов, которым присваивается нулевое значение полю number и выводится сообщение о создании объекта суперкласса с указанием значения поля.

Подкласс MyClassB, создаваемый на основе суперкласса MyClassA, также содержит описание числового поля number. Описанный в классе метод showB() выводит на экран значение поля number, а конструктор без аргументов позволяет создать объект подкласса с полем number, инициализированным по умолчанию значением 100. Таким образом, в программном коде класса MyClassB складывается довольно интересная ситуация: класс имеет два поля number. Объявленное непосредственно в классе поле «перекрывает» наследу-

емое поле с таким же именем, поэтому как в методе showB(), так и в конструкторе подкласса инструкция number является обращением именно к полю, описанному в классе.

В главном методе main() в классе TwoFieldsDemo создается объект obj подкласса MyClassB. Результатом выполнения команды new MyClassB() являются сообщения:

Создан объект суперкласса с полем 0
Создан объект подкласса с полем 100

Первое сообщение появляется в результате вызова конструктора суперкласса в рамках вызова конструктора подкласса. Конструктор суперкласса «своему» полю number присваивает значение 0 и выводит сообщение о создании объекта суперкласса. Затем выполняются команды из тела конструктора подкласса.

В результате другому полю number (описанному в подклассе) присваивается значение 100 и выводится сообщение о создании объекта подкласса. Таким образом, при создании поля number объекта obj получают значения 0 и 100.

В главном методе при обращении к полю number командой obj.number=50 изменяется значение того поля, которое описано в подклассе. Другими словами, поле number, имевшее значение 100, получает значение 50.

При выводе значения поля number командой obj.showA() выполняется обращение к полю, описанному в суперклассе: метод showA() обращается в своем программном коде к полю по имени и для него это то поле, которое описано в суперклассе — там же, где описан соответствующий метод. Командой obj.showB() выводится значение поля number, описанного в подклассе.

Чтобы различать одноименные поля, описанные и унаследованные, указывают инструкцию super, то есть ту же самую инструкцию, что и при вызове конструктора суперкласса. Только в этом случае синтаксис ее использования несколько иной.

Обращение к полю, наследованному из суперкласса (описанному в суперклассе), выполняется в формате super.имя_поля. Например, чтобы в методе showB() из рассмотренного примера обратиться к полю number суперкласса, достаточно воспользоваться инструкцией super.number. В листинге приведен измененный код предыдущего примера, в котором в подклассе выполняется обращение как к унаследованному, так и описанному непосредственно в подклассе полю number:

```
// Суперкласс:
class MyClassA{
// Поле:
int number;
// Конструктор суперкласса:
MyClassA(int a){
number=a;
System.out.println("Создан объект суперкласса с полем "+number);}
// Отображение значения поля:
void showA(){
System.out.println("Поле number: "+number);}
}
// Подкласс:
```

```
class MyClassB extends MyClassA{
// Поле с тем же именем:
int number;
// Конструктор подкласса:
MyClassB(int a){
super(a-1); // Вызов конструктора суперкласса
number=a; // Поле из подкласса
// Обращение к полю из суперкласса и подкласса:
System.out.println("Создан объект с полями: "+super.number+" и
"+number);}
// Отображение значения поля:
void showB(){
// Обращение к полю из суперкласса и подкласса:
System.out.println("Поля объекта "+super.number+" и "+number);}
}
class TwoFieldsDemo2{
public static void main(String[] args){
// Создание объекта подкласса:
MyClassB obj=new MyClassB(5);
// Изменение значения поля:
obj.number=10;
// Отображение значений полей:
obj.showA();
obj.showB();
}}
```

В отличие от предыдущего случая, конструктору суперкласса передается аргумент, который присваивается в качестве значения полю number. Как и ранее, значение поля отображается с помощью метода суперкласса showA().

Конструктор подкласса также имеет аргумент. Значение аргумента присваивается полю number, определенному непосредственно в классе. Одноименное наследуемое поле получает значение, на единицу меньшее аргумента конструктора.

Для этого вызывается конструктор суперкласса с соответствующим аргументом.

При выполнении конструктора подкласса также выводится сообщение о значении двух полей, причем обращение к полю, определенному в подклассе, выполняется по имени number, а обращение к полю, определенному в суперклассе, через инструкцию super.number. Значения обоих полей можно вывести на экран с помощью метода showB().

Главный метод программы содержит команду создания объекта подкласса, команду изменения значения поля number, определенного в подклассе (инструкцией obj.number=10), а также команды вывода значений полей с помощью методов showA() и showB(). В результате выполнения этой программы получаем следующее:

Создан объект суперкласса с полем 4
Создан объект с полями: 4 и 5
Поле number: 4
Поля объекта 4 и 10

По тому же принципу, что и замещение полей с совпадающими именами, замещаются и методы с одинаковыми сигнатурами. Однако с методами ситуация обстоит несколько сложнее, поскольку существует такой механизм,

как перегрузка методов. Кроме перегрузки важным понятием является переопределение методов.

Переопределение методов при наследовании

Как уже отмечалось, если в подклассе описан метод с сигнатурой, совпадающей с сигнатурой метода, наследуемого из суперкласса, то метод подкласса замещает метод суперкласса. Другими словами, если вызывается соответствующий метод, то используется та его версия, которая описана непосредственно в подклассе. При этом старый метод из суперкласса становится доступным, если к нему обратиться в формате ссылки с использованием ключевого слова super.

Между переопределением и перегрузкой методов существует принципиальное различие. При перегрузке методы имеют одинаковые названия, но разные сигнатуры. При переопределении совпадают не только названия методов, но и полностью сигнатуры (тип результата, имя и список аргументов). Переопределение реализуется при наследовании. Для перегрузки в наследовании необходимости нет. Если наследуется перегруженный метод, то переопределение выполняется для каждой его версии в отдельности, причем переопределяются только те версии перегруженного метода, которые описаны в подклассе. Если в подклассе какая-то версия перегруженного метода не описана, эта версия наследуется из суперкласса.

Может сложиться и более хитрая ситуация. Допустим, в суперклассе определен некий метод, а в подклассе определяется метод с таким же именем, но другой сигнатурой. В этом случае в подклассе будут доступны обе версии метода: и исходная версия, описанная в суперклассе, и версия метода, описанная в подклассе.

То есть имеет место перегрузка метода, причем одна версия метода описана в суперклассе, а вторая — в подклассе.

В листинге приведен пример программы с кодом переопределения метода:

```
class ClassA{
static int count=0;
private int code;
int number;
ClassA(int n){
set(n);
count++;
code=count;
System.out.println("Объект №"+code+" создан!");}
void set(int n){
number=n;}
void show(){
System.out.println("Для объекта №"+code+":");
System.out.println("Поле number: "+number);}
}
class ClassB extends ClassA{
char symbol;
ClassB(int n,char s){
super(n);
symbol=s;}
void set(int n,char s){
```

```
number=n;
symbol=s;}
void show(){
super.show();
System.out.println("Поле symbol: "+symbol);}
}
class MyMethDemo{
public static void main(String[] args){
ClassA objA=new ClassA(10);
ClassB objB=new ClassB(-20,'a');
objA.show();
objB.show();
objB.set(100);
objB.show();
objB.set(0,'z');
objB.show();}
}
```

В результате выполнения программы получаем последовательность сообщений:

Объект №1 создан!
Объект №2 создан!
Для объекта №1:
Поле number: 10
Для объекта №2:
Поле number: -20
Поле symbol: a
Для объекта №2:
Поле number: 100
Поле symbol: a
Для объекта №2:
Поле number: 0
Поле symbol: z

Разберем программный код и результат его выполнения. В программе описывается класс ClassA (суперкласс), на основе которого создается подкласс ClassB.

Класс ClassA имеет целочисленное поле number, статическое целочисленное поле count (инициализированное нулевым значением) и закрытое целочисленное поле code. Кроме этого, в классе описан конструктор с одним аргументом (значением поля number), метод set() с одним аргументом для присваивания значения полю number, а также метод show() для отображения значения поля number.

Статическое поле count предназначено для учета количества созданных объектов.

При создании очередного объекта класса значение этого счетчика увеличивается на единицу. Для этого в конструкторе класса ClassA размещена команда count++.

Кроме этого в конструкторе с помощью метода set() присваивается значение полю number (в качестве аргумента методу передается аргумент конструктора), а командой code=count присваивается значение закрытому полю code. В поле code записывается порядковый номер, под которым создан соответствующий объект.

Поле count для этой цели не подходит, поскольку оно статическое и изменяется каждый раз при создании очередного объекта. В поле code записывается значение поля count после создания объекта и впоследствии поле code этого объекта не меняется.

Поле code (после присваивания значения полю) служит в конструкторе для вывода сообщения о создании объекта с соответствующим номером. Номер объекта (поле code) используется также в методе show(), чтобы легче было проследить, для какого именно объекта выводится информация о значении поля number.

Подкласс ClassB создается на основе суперкласса ClassA. В подклассе ClassB наследуется статическое поле count и поле number. Закрытое поле code не наследуется. Кроме этих наследуемых полей, непосредственно в классе ClassB описано символьное поле symbol. Конструктор класса принимает два аргумента: первый типа int для поля number и второй типа char для поля symbol.

Код конструктора класса ClassB состоит всего из двух команд: команды вызова конструктора суперкласса super(n) и команды присваивания значения символьному полю symbol=s (n и s — аргументы конструктора). Со второй командой все просто и понятно. Интерес представляет команда вызова конструктора суперкласса. Во-первых, этим конструктором наследуемому полю number присваивается значение. Во-вторых, значение наследуемого статического поля count увеличивается на единицу. Это означает, что ведется общий учет всех объектов, как суперкласса, так и подкласса. В-третьих, хотя поле code не наследуется, под него выделяется место в памяти и туда заносится порядковый номер созданного объекта. На экран выводится сообщение о создании нового объекта, а номер объекта считывается из «несуществующего» поля code.

Метод show() в классе ClassB переопределяется. Сигнатура описанного в классе ClassB метода show() совпадает с сигнатурой метода show(), описанного в классе ClassA. Если в классе ClassA методом show() отображается информация о номере объекта и значении его поля number, то в классе ClassB метод show() выводит еще и значение поля symbol. При этом в переопределенном методе show() вызывается также прежняя (исходная) версия метода из класса ClassA. Для этого используется инструкция вида super.show(). Этот исходный вариант метода, кроме прочего, считывает из ненаследуемого (но реально существующего) поля code порядковый номер объекта и отображает его в выводимом на экран сообщении.

Метод set() в классе ClassB перегружается. Хотя в классе ClassA есть метод с таким же названием, сигнатуры методов в суперклассе и подклассе разные.

В суперклассе у метода set() один числовой аргумент, а в подклассе у этого метода два аргумента: числовой и символьный. Поэтому в классе ClassB имеется два варианта метода set() — с одним и двумя аргументами. Первый наследуется из суперкласса ClassA, а второй определен непосредственно в подклассе ClassB.

В главном методе программы командами ClassA objA=new ClassA(10) и ClassB objB=new ClassB(-20,'a') создаются два объекта: объект objA суперкласса и объект objB подкласса. В результате выполнения этих команд на экране появляются сообщения Объект №1 создан! и Объект №2 создан! — сообщения выводятся конструкторами. Проверяются значения полей созданных объектов командами objA.show() и objB.show(). Поскольку метод show() перегружен, то в первом случае вызывается метод show(), описанный в суперклассе ClassA, а во втором — метод show(), описанный в подклассе ClassB. Поэтому для объекта objA выводится значение одного (и единственного) поля, а для объекта objB — значения двух полей.

Командой objB.set(100) метод set() вызывается из объекта objB. Поскольку в данном случае методу передан всего один аргумент, вызывается версия метода, описанная в классе ClassA. В результате меняется значение поля number объекта objB, а поле symbol остается неизменным. Подтверждается данное утверждение после вызова метода objB.show() (см. приведенный ранее результат выполнения программы). Если же воспользоваться командой objB.set(0,'z'), будет вызван тот вариант метода set(), который описан в классе ClassB. Выполнение команды objB.show() показывает, что в результате изменились оба поля объекта objB.