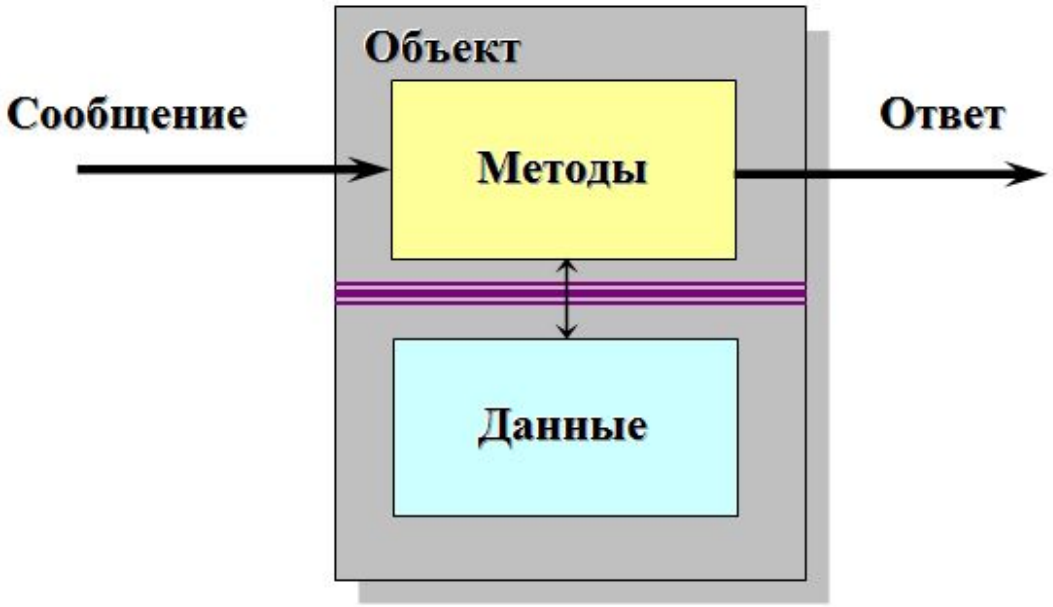
Об'єктно-орієнтований підхід в Java

Объекты программирования имеют две характеристики: состояние и поведение: состояние объекта определяется его свойствами, а поведение – выполняемыми им действиями.

Для объектно-ориентированных языков характерны следующие подходы к программированию:

- все является объектом;
- программа является набором объектов, указывающих друг другу, что делать посредством посылки сообщений;
- каждый объект имеет некоторый тип;
- все объекты одного типа могут получать одни и те же сообщения.

Объект – это набор данных и процедур, работающих с этими данными. Эти процедуры (подпрограммы) обработки данных объекта называются в Java **методами**.



Методы объекта имеют полный доступ к данным своего объекта. Вне объекта данные, объявляемые в данном объекте, не видны, а все взаимодействие с объектом осуществляется только через методы.

Объект может быть разделен на две компоненты: внешнюю и внутреннюю. Внешнюю часть (**интерфейс**) составляют методы, осуществляющие взаимодействие с остальной частью программы. Внутреннюю часть составляют данные и методы, доступные только внутри объекта.

Инкапсуляция – это процесс упаковки данных объекта вместе с его методами. Результатом инкапсуляции является предотвращение нежелательного доступа извне к данным и методам внутри объекта и возможность изменения внутренней реализации объекта без изменения других частей программы.

Объекты программы взаимодействуют и связываются друг с другом с помощью **сообщений**.

Информация, необходимая объекту для выполнения тех или иных методов данного объекта передается в сообщении с помощью **параметров**.

Таким образом, **сообщение должно содержать три компонента**:

- объекта – получателя сообщения;
- имени выполняемого действия;
- параметров, необходимых для выполнения действия.

Интерфейсы позволяют объектам передавать и принимать сообщения, даже если они расположены в разных узлах сети

Чтобы определить объект, в Java и других объектно-ориентированных языках используется понятие: **класс**.

Класс – это шаблон или прототип, определяющий тип объекта. Класс содержит описание переменных и констант, характеризующих свойства объекта. Они называются **полями класса** (class fields). Процедуры, описывающие поведение объекта, называются **методами класса** (class methods).

Когда объект создается из класса, переменные, объявленные для этого класса, размещаются в памяти. Затем переменные могут модифицироваться с помощью методов объекта. Реализации одного класса совместно используют реализации методов класса, но каждая из них использует свои собственные данные объекта, т.е. класс можно **повторно использовать** для реализации нескольких объектов с одинаковыми методами, но разными значениями данных.

Оголошення класу в Java:

```
class идентификатор-класса
{
    тело-класса
}
```

идентификатор определяет имя класса.

В теле класса (и только в теле класса) определяются его переменные и методы.

Программа на языке Java представляет собой объявления одного или нескольких классов. Каждый класс в программе компилируется в отдельный файл с именем ***идентификатор-класса*.class**.

```
class Circle
{
…
}
```

Приклади створення об'єктів класу Circle

```
Circle obj1;
```
- создание переменной obj1 типа Circle

```
obj1 = new Circle();
```
- создается новый объект типа Circle, ссылка на него (адрес объекта) записывается в переменную obj1

```
Circle obj1 = new Circle();
```
- совмещенное задание объектной переменной и назначение ей объекта

```
Circle obj1 = new Circle(130,120,50);
```
- создание переменной со списком параметров

```
Circle circle1 = new Circle(130,120,50);
Circle circle2 = new Circle(130,120,50);
```
- создание двух независимых объектов одного класса с одинаковыми начальными параметрами

```
obj1.x = 5;
```
- обращение к полю данных **x** объектной переменной obj1

```
obj1.show();
```
- обращение методу **show()** объектной переменной obj1

Методи в Java

Определение метода:

```
возвращаемый-тип идентификатор-метода (параметры)
{
    тело-метода
}
```

Возвращаемый-тип определяет тип данных, которые возвращает метод при вызове.
Идентификатор-метода определяет имя метода, а ***параметры*** – список параметров, которые необходимо передать методу при его вызове.
Тело-метода содержит операторы, реализующие действия, выполняемые данным методом.

Если тип возвращаемого значения не **void**, **в теле метода должен быть хотя бы один оператор**

```
return выражение;
```

Пример

Определение метода:

```
int sumOfTwoValues(int a, int b)
{
    int x;
    x = a + b;
    return x;
}
```

Вызов метода

```
;int y
y=sumOfTwoValues(5, 3); // y = 8
```

В языке Java в пределах одного класса можно определить два или более методов, которые совместно используют одно и то же имя, но имеют разное количество параметров. Такие методы называют **перегруженными**, а о процессе говорят как о **перегрузке метода** (method overloading).

В Java можно использовать методы, реализация которых, выполнена во внешнем файле, в программе написанном на языке C/C++. Для этого перед определением метода (но без тела метода) указывается модификатор **native**

Змінні типу класів в Java

```
MyClass obj1;
MyClass obj2;
```

- две переменных класса **MyClass** с именами **obj1 и obj2**

Для объявленной переменной типа класса необходимо создать **объект, экземпляр** (instance) описанного класса.

Когда создается объект, необходимо инициализировать его переменные. Для этого в классе определяется специальный метод (перегруженные методы), имя которого совпадает с именем класса. Эти методы называют **конструкторами.**

Конструктор отличается от обычных методов следующими основными особенностями:

- конструктор не должен возвращать никакого значения;
- тип возвращаемого значения для конструктора не указывается

Если конструктор в классе не определен, компилятор Java создает конструктор по умолчанию.

Создание объекта

идентификатор-переменной **= new** *идентификатор-конструктора*(***параметры***);

идентификатор-переменной – имя создаваемого объекта

идентификатор-конструктора – имя вызываемого конструктора класса

параметры – список параметров, передаваемых конструктору класса

```
obj1 = new MyClass();
obj2 = new MyClass(12, 5);
```

Операции объявления и инициализации переменных типа класса могут быть объединены в одном операторе

```
MyClass obj3 = new MyClass(8, 4);
```

Змінні і методи об'єкта

Обращение к переменным и вызов методов объекта

имя-объекта.***имя-переменной***
и
имя-объекта.***имя-метода*(*аргументы*)**

имя-объекта – это идентификатор переменной объекта класса

имя-переменной – идентификатор переменной класса

имя-метода – идентификатор метода класса

аргументы – значения, задаваемые при вызове метода

```
int var1InObj1 = obj1.var1;
```

```
obj2.var1 = 12;
```

```
obj1.setVar1(2);
obj2.setVar1(2);
```

Если имя объекта для переменной или метода не задано, то компилятор Java считает, что данная переменная или метод определены в данном классе. Можно в этом случае вместо имени объекта указать ключевое слово **this**.

Обычно такое указание используется в тех случаях, если имя переменной класса и аргумент метода в классе совпадают

```
class MyClass
{
      int var1;
      …
      void setVar1(int var1)
      {
            this.var1=var1;
      }
}
```

Змінні і методи класу

Модификатор **static** задает переменную, общую для всех объектов данного класса.

Для работы со статическими переменными обычно создаются статические методы, помеченные модификатором **static.**
Статические методы и переменные называют также **методами и переменными класса** (class variables and methods), поскольку к ним можно обращаться, указывая не имя объекта, а имя класса.

Пример

Определение

```
static int var2 = 0;
…
static void setVar2(int var)
{
      var2 = var;
}
```

Обращение

```
int val2Value = MyClass.var2;
MyClass.setVar2(12);
```

Основная особенность статических переменных и методов: доступ к ним выполняется, даже если не создан ни один экземпляр класса.

Для статических методов действуют следующие основные ограничения:

- в статическом методе нельзя использовать ссылки **this;**
- в статическом методе нельзя обращаться к нестатическим переменным (все переменные, объявленные вне статического метода и используемые внутри него, должны быть объявлены с модификатором **static);**
- в статическом методе нельзя прямо вызывать нестатические методы (все методы, вызываемые из статического метода, должны быть объявлены с модификатором **static).**

Операції над об'єктами

присваивание "=" – присвоение указателя на объект ссылочной переменной (при этом новой копии объекта не создается);

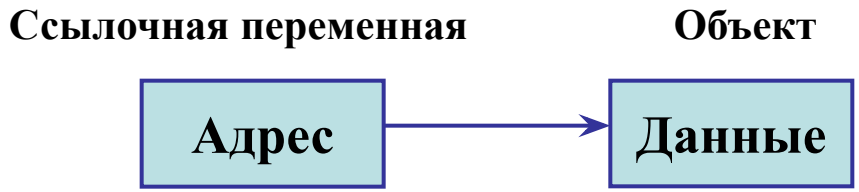
проверка на равенство "==" и на неравенство "!=" – результатом этих операций будет **true** или **false**, в зависимости от того, указывают ли сравниваемые переменные на один и тот же объект в памяти.

Операция *имя-объекта* **instanceof** *имя-класса*

Результатом этой операции является **true**, если объект с идентификатором ***имя-объекта*** является реализацией класса с идентификатором ***имя-класса***, и **false** – в противном случае.

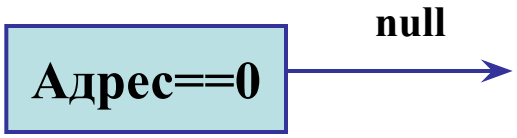
```
MyClass obj4 = new MyClass(15, 7);
MyClass obj5 = new MyClass(15, 7);
MyClass obj6 = obj4;
boolean isEqual1, isEqual2, isInstance;
isEqual = obj6 == obj4;                  // isEqual1 - true
isEqual2 = obj4 == obj5;                 // isEqual2 - false
isInstance = obj4 instanceof MyClass;    // isInstance - true
```

Робота зі посилальними змінними



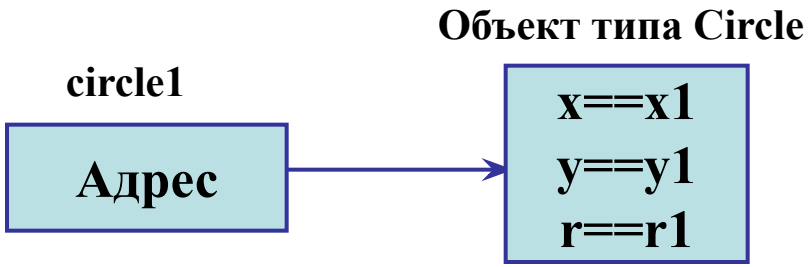
Ссылочная переменная и связанный с ней объект

Переменная circle1 типа Circle



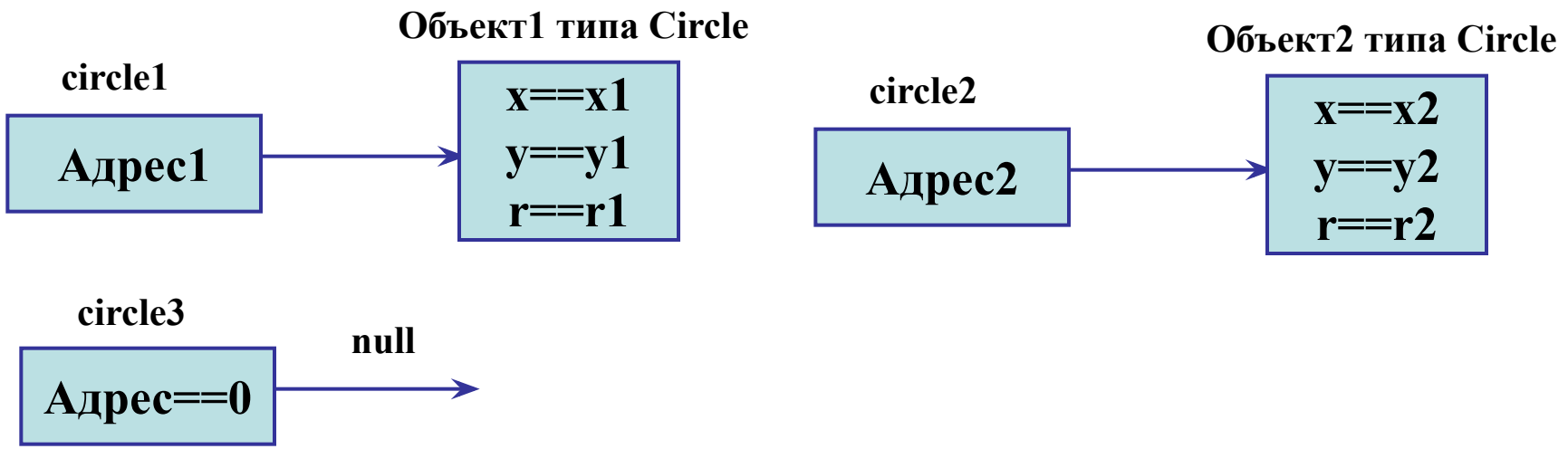
Начальное состояние ссылочной переменной circle1

```
circle1 = new Circle(x1, y1, r1);
```

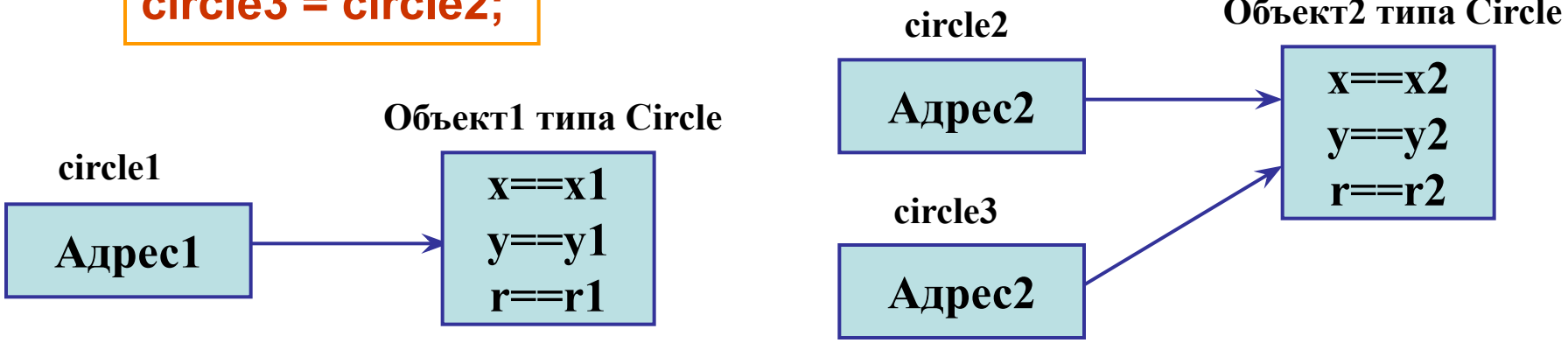


Ссылочная переменная **circle1** и связанный с ней объект типа **Circle**

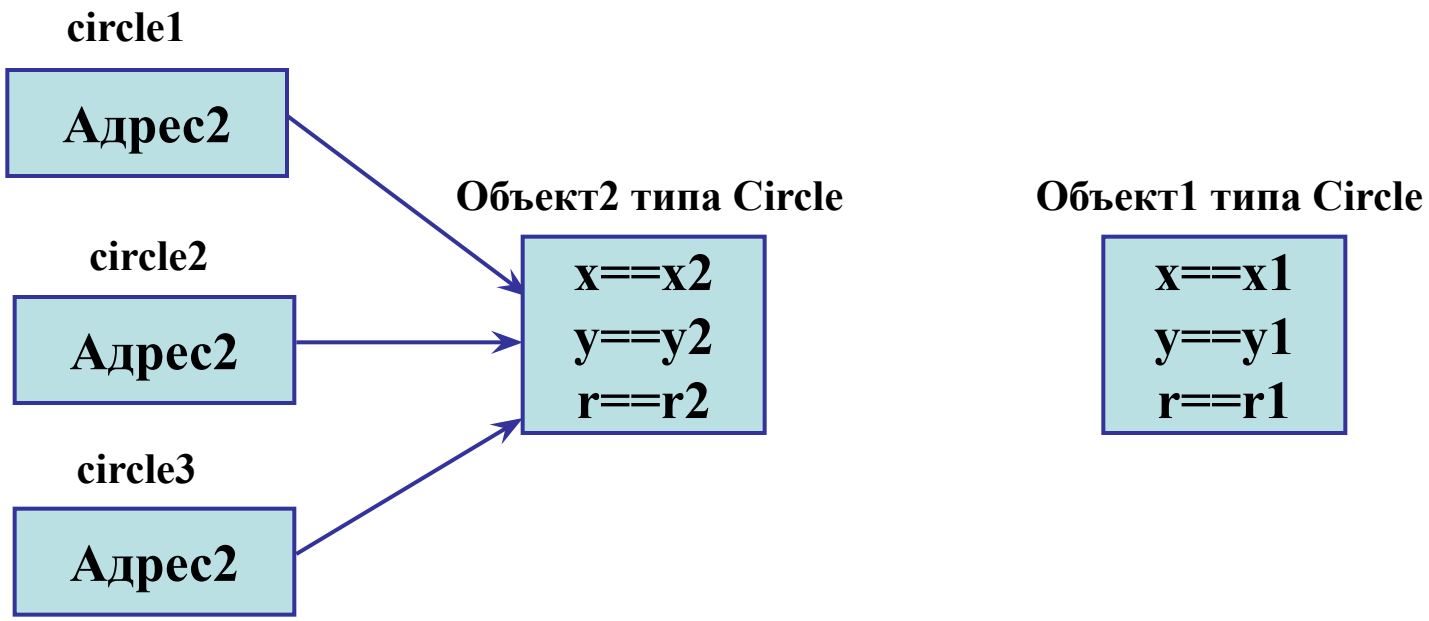
```
Circle circle1 = new Circle(x1, y1, r1);
Circle circle2 = new Circle(x2, y2, r2);
Circle circle3;
```




```
circle3 = circle2;
```



```
circle1 = circle2;
```



Об'єктні надбудови примітивних типів

Многие классы в Java работают не с примитивными типами данных, а с объектами, поскольку для объектов можно задавать свойства и методы.

С этой целью в Java введены классы – **объектные надстройки над примитивными типами**. Новые объекты, например, числа в этих классах задаются с помощью оператора **new** и над ними нельзя выполнять операции, определенные для примитивных типов (например, сложение), однако они часто используются для выполнения некоторых операций, реализуемых с помощью методов соответствующего класса-надстройки.

Все классы-надстройки автоматически доступны программе, поскольку они находятся в пакете java.lang.

Оболонкові класи.
Упаковка (boxing) і розпакування (unboxing)

Примитивный тип	Оболочечный класс
byte	Byte
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double

```
Integer obj1 = 10;   //упаковка
int i1 = obj1*2;     //распаковка при вычислении выражения
```

```
Byte b = 1;       //упаковка
obj1 = i1/10;         //упаковка
b = 2;            //упаковка
```

Пакети

Все классы Java распределены по **пакетам** (обычно по функциональному признаку, например, классы-утилиты или классы ввода-вывода). Кроме классов, пакеты могут включать в себя **интерфейсы** и вложенные **подпакеты** (subpackages). В результате образуется древовидная структура пакетов и подпакетов, которая соответствует структуре файловой системы. Все файлы с расширением **.class**, образующие пакет, хранятся в одном каталоге, а подпакеты хранятся в подкаталогах этого каталога.

Пакет однозначно идентифицируется своим именем, перед которым, отделенные друг от друга точкой, идут имена всех пакетов, находящихся выше данного пакета в уровнях иерархии.

java.awt.event обозначает имя подпакета **event**, находящегося в подпакете **awt**, который находится в пакете **java**

javax.swing.event обозначает имя другого подпакета с тем же именем **event**, но который находится в подпакете **swing** пакета **javax**.

Пакети

```
package имя_пакета;
```

чтобы поместить класс в пакет, требуется продекларировать имя пакета в начале файла, в котором объявлен класс

Также необходимо поместить исходный код класса соответствующую папку

```
package pkg1.pkg2.pkg3;
class MyClass1 {
....
}
```

Вложенным пакетам соответствуют составные имена

```
public class MyClass2 {
....
}
```

При декларации класса можно указывать, что он общедоступен, с помощью *модификатора доступа* public.

При этом возможен доступ к данному классу из других пакетов.
Если же модификатор public отсутствует, то доступ к классу разрешен только из классов, находящихся с ним в одном пакете.

Все имена классов, интерфейсов и подпакетов в пакете должны быть уникальны.

В Java принято имена пакетов писать только строчными буквами, имена классов начинать с заглавной буквы, а имена полей данных и методов начинать со строчной буквы. Если имя класса, поля данных или метода состоит из нескольких слов, то каждое новое слово принято писать с заглавной буквы.

```
javax.swing.JMenuItem
```

javax и **swing** — пакеты,
JMenuItem — имя класса.

При компиляции необходимые для выполнения программы классы пакетов Java, за исключением пакета **java.lang**, автоматически не включаются. Чтобы сделать их доступными в программе, можно либо указывать полное имя класса, либо использовать оператор **import** с именем пакета и именем используемого класса данного пакета:

```
import java.util.Date;
```

Если необходимо использовать несколько классов из пакета, обычно вместо имени класса ставится символ **"*"**, что указывает, что данной программе будут доступны **все** классы и интерфейсы данного пакета.

```
import java.awt.*;
```

- делает доступными программе все классы из пакета **java.awt**.

Все операторы import принято располагать в самом начале программы

Импортируются только имена файлов, находящихся на уровне указанного пакета. Импорта имен из вложенных в него пакетов не происходит.

```
import pkg1.pkg2.pkg3.MyClass2;
import pkg1.pkg2.pkg3.*;
```

MyClass2 импортирован

```
import pkg1.*;
import pkg1.pkg2.*;
```

MyClass2 не импортирован

Пример

Объявить графический объект **g**, имеющий тип **java.awt.Graphics**, можно тремя способами:

1) напрямую с указанием имени пакета и класса:

```
java.awt.Graphics g;
```

2) с предварительным импортом класса **Graphics** из пакета **java.awt** и последующим указанием имени этого класса без его спецификации именем пакета:

```
import java.awt.Graphics;
...
Graphics g;
```

3) с предварительным импортом всех классов из пакета **java.awt** и последующим указанием имени этого класса без его спецификации именем пакета:

```
import java.awt.*;
...
Graphics g;
```

Обращение к переменным класса и методам класса должно идти только через имя класса

java.lang.Math

$$y = \frac{\sin \pi x}{\pi x}$$

```
y=Math.sin(Math.PI*x)/(Math.PI*x);
```

Статический импорт из пакета классов переменных класса и методов класса.

```
import static java.lang.Math.PI;
import static java.lang.Math.sin;
```

или

```
import static java.lang.Math.*;
```

```
y = sin(PI*x)/(PI*x);
```

Базові пакети і класи Java

Вміст пакету java

java.applet Поддержка работы с апплетами

java.awt Базовый пакет работы с графическим пользовательским интерфейсом (Abstract Window Toolkit — Абстрактный инструментарий графического окна)

java.beans Поддержка компонентной модели JavaBeans

java.io Поддержка базовых средств ввода/вывода

java.lang Содержит базовые классы языка Java. Автоматически импортируется в любую программу без указания имени пакета

java.lang.reflect Поддерживает механизм доступа к классам как метаобъектам, обеспечивает динамическое выяснение программой, какие возможности поддерживает класс. Данный механизм называется reflection - "отражение"

java.lang.Math Класс, обеспечивающий поддержку основных математических функций, а также простейшее средство генерации псевдослучайных чисел

java.math Поддержка вычислений с целыми числами произвольной длины, а также числами в формате с плавающей точкой произвольной точности

java.net Поддержка работы в Интернете, а также соединений через сокеты (sockets)

java.nio Содержит классы и пакеты для поддержки сетевых соединений, расширяющие возможности пакета java.io. В частности, содержит классы контейнеров (буферов) для создания списков с данными различных примитивных типов, а также пакеты channels (каналы соединения, коннекции) и charset (национальный набор символов). Пакет charset обеспечивает поддержку перекодирования из символов Unicode в последовательность байтов для передачи через канал связи, а также обратное преобразование

java.rmi Поддержка вызовов удаленных методов

java.security Поддержка специальных средств, обеспечивающих безопасность приложения, в том числе при работе в компьютерных сетях (списки доступа, сертификаты безопасности, шифрование и т.д.)

java.sql Поддержка SQL-запросов к базам данных

java.text Поддержка специальных средств, обеспечивающих локализацию программ, — классы, поддерживающие работу с текстом, датами, текстовым представлением чисел. Кроме того, содержит средства зависимого от локализации сравнения строк

java.util Содержит важнейшие классы для работы со структурами данных (в том числе необходимых для работы с событиями и датами). В частности — поддержку работы с массивами (сортировка, поиск), а также расширенные средства генерации псевдослучайных чисел

java.util.jar Поддержка работы с jar-архивами (базовым видом архивов в Java)

java.util.zip Поддержка работы с zip-архивами

Вміст пакету javax

поддержка возможностей, появившихся в Java 2

javax.accessibility Обеспечивает настройку специальных возможностей представления информации для людей с плохим зрением, слухом и т. п., а также других случаев, когда требуется специализированный доступ для управления информационными объектами

javax.activity Вспомогательный пакет для работы с компонентами

javax.crypto Поддержка шифрования-расшифровки данных

javax.imageio Поддержка работы с изображениями (ввод/вывод)

javax.management Поддержка работы с управляющими компонентами (MBean — Management Bean)

javax.naming Поддержка работы с пространством имен компонентов

javax.net Поддержка работы в Интернете, а также соединений через сокеты (sockets). Расширение возможностей java.net

javax.print Поддержка работы с печатью документов

javax.rmi Поддержка вызовов удаленных методов. Расширение возможностей java.rmi

javax.security Поддержка специальных средств, обеспечивающих безопасность приложения. Расширение возможностей java.security

javax.sound Поддержка работы со звуковыми потоками и файлами

javax.sql Поддержка SQL-запросов к базам данных. Расширение возможностей java.sql

javax.swing Библиотека основных графических компонентов в Java2

javax.transaction Поддержка работы с транзакциями

javax.xml Поддержка работы с XML-документами и парсерами

Вміст пакету org

пакеты, предоставляемые свободным сообществом разработчиков

org.ietf Поддержка защищенных соединений по протоколу GSS (Kerberos v5)

org.omg Средства для использования из программ на Java технологии CORBA, применяемой для создания распределенных объектных приложений

org.w3c Интерфейсы для работы с XML-документами в соответствии со спецификацией DOM

org.xml Поддержка работы с XML-документами

Вміст пакету com.sun

обеспечивает расширение возможностей пакета javax

com.sun.accessibility Дополнение к пакету javax.accessibility

com.sun.beans Дополнение к пакету java.beans

com.sun.corba Поддержка работы в компьютерных сетях с базами данных по технологии CORBA (Common Object Request Broker Architecture)

com.sun.crypto Дополнение к пакету javax.crypto

com.sun.image Поддержка работы с изображениями

com.sun.imageio Дополнение к пакету javax.imageio

com.sun.java Поддержка стилей показа приложений, а также служебные утилиты для работы с браузерами и WWW-документами

com.sun.java_cup Поддержка технологии JavaCup

com.sun.jlex Поддержка работы лексического анализатора

com.sun.jmx Дополнение к пакету javax.management

com.sun.management Дополнение к пакету javax.management

com.sun.media Поддержка работы со звуком

com.sun.naming Дополнение к пакету javax.naming

com.sun.net Дополнение к пакету javax.net

com.sun.org Поддержка взаимодействия с сервером Apache, средства работы с базами данных по технологии CORBA

com.sun.rmi Дополнение к пакету javax.rmi

Спадкування і поліморфізм

Наследование позволяет строить на основе первоначального класса другие, добавляя в классы новые поля данных и методы.

Первоначальный класс называется ***прародителем*** (ancestor), новые классы — его ***потомками*** (descendants).

Набор классов, связанных отношением наследования, называется ***иерархией классов***.

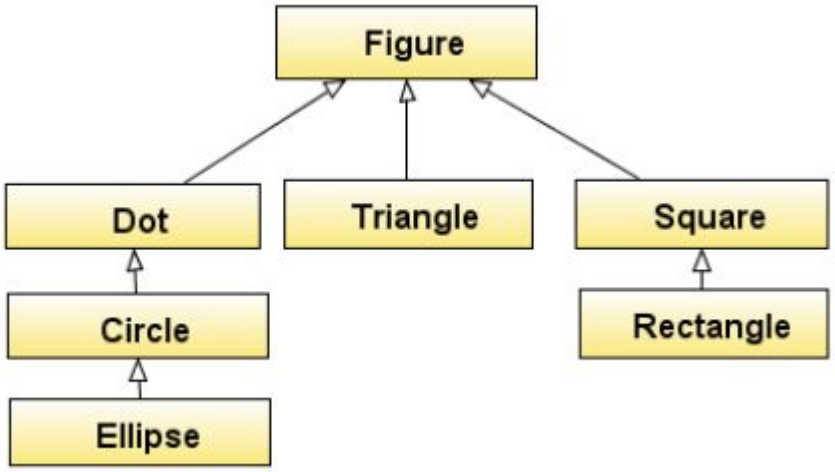
Класс, стоящий во главе иерархии, от которого унаследованы все остальные (прямо или опосредованно), называется ***базовым классом иерархии***.

Полиморфизм (с греческого «имеющий много форм») — наличие кода, написанного с использованием ссылок, имеющих тип базового класса иерархии.

При этом такой код должен правильно работать для любого объекта, являющегося экземпляром класса из данной иерархии, независимо от того, где этот класс расположен в иерархии. Такой код и называется ***полиморфным***. При написании полиморфного кода заранее неизвестно, для объектов какого типа он будет работать — один и тот же метод будет исполняться по-разному в зависимости от типа объекта.

Преимущество объектного программирования заключается в возможности написания полиморфного кода. Именно для этого создается иерархия классов. Полиморфизм позволяет резко увеличить ***коэффициент повторного использования*** программного кода и ***модифицируемость*** этого кода по сравнению с процедурным программированием.

Приклад ієрархії класів для зображення фігур на екрані



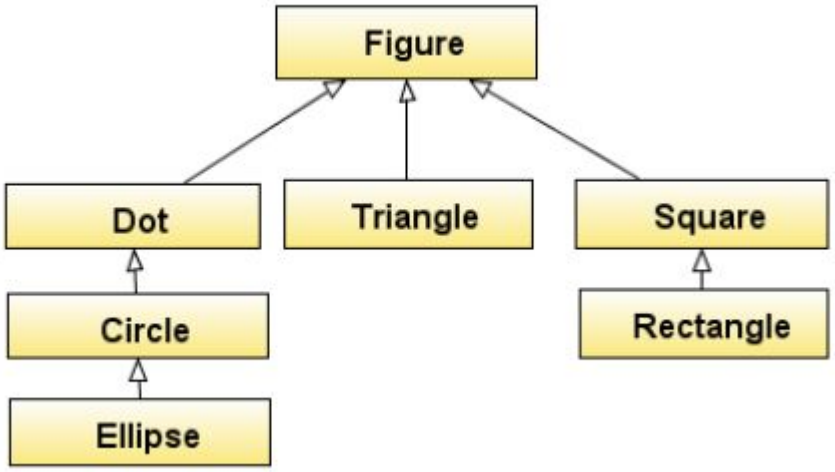
Класс **Figure**: поля данных ***x*** и ***y*** — координаты фигуры на экране.

Класс **Dot**: поля данных ***x*** и *y*, наследуемые от **Figure**. В самом классе **Dot** задавать эти поля не надо

Класс **Circle**: поля ***x*** и ***y***, наследуемые от **Figure**,
новое поле r, соответствующее радиусу, новый метод ***setSize***,
обеспечивающий изменение радиуса

Класс **Ellipse**: поля ***x*** и ***y***, наследуемые от Figure, поле ***r*** и метод ***setSize***, наследуемые от ***Circle***, новое поле данных ***r2***, задающее длину второй полуоси эллипса

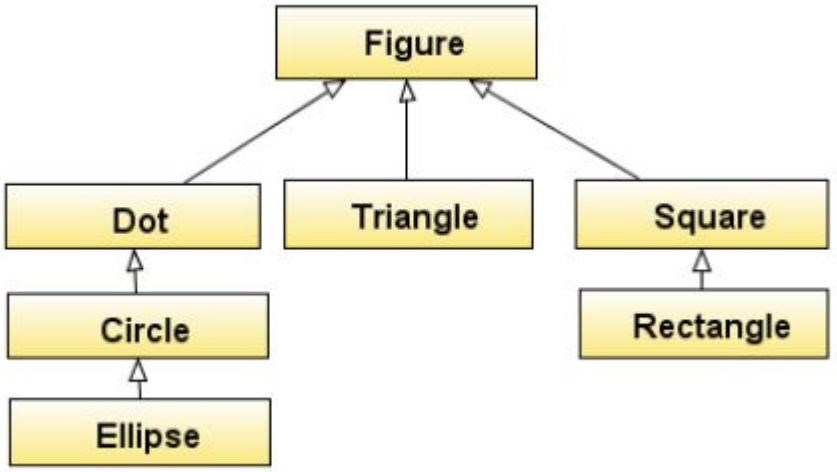
Приклад ієрархії класів для зображення фігур на екрані



Класс **Square**: поля данных ***x*** и ***y***, наследуемые от **Figure**, новое поле ***a***, соответствующее стороне квадрата.

Класс **Rectangle**: поля данных ***x*** и ***y***, наследуемые от **Figure**, поле ***a***, наследуемое от **Square,** новое поле ***b***, соответствующее стороне прямоугольника.

Приклад ієрархії класів для зображення фігур на екрані



Класс **Triangle**: поля данных ***x*** и ***y***, наследуемые от **Figure,** в качестве новых, не унаследованных полей данных, могут выступать либо координаты вершин треугольника, либо координаты одной из вершин, либо длины прилегающих к ней сторон и угол между ними и т. д.

Потомки должны обладать более сложным устройством и поведением по сравнению с прародителем.

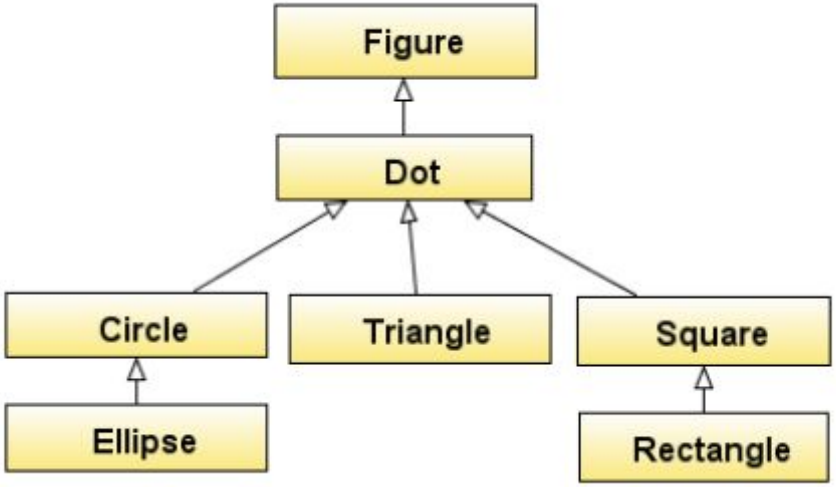
Каждый объект класса-потомка ***при любых значениях полей*** нужно рассматривать как экземпляр класса-прародителя.

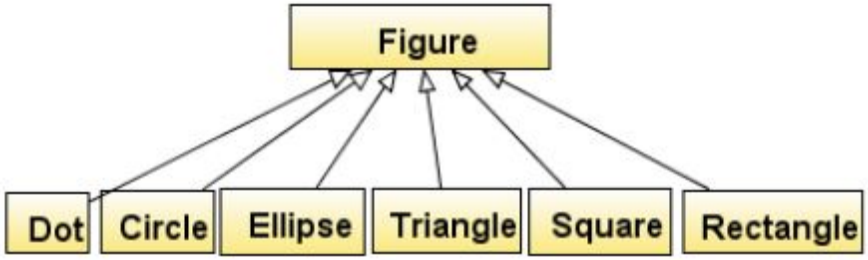
По своему поведению объект-эллипс может рассматриваться как любой экземпляр типа **Circle** и даже вести себя в точности как окружность, но не наоборот: объект типа **Circle** не обладает поведением **Ellipse**.

КРИТЕРІЙ ЗАСТОСОВНОСТІ УСПАДКУВАННЯ

Если имеются классы **A1** и **A2** и можно считать, что **A2** является модифицированным (усложненным или измененным) вариантом **A1** с сохранением всех особенностей поведения **A1**, то **A2** должен описываться как потомок **A1**. На уровне абстракции, описывающей поведение, объект типа **A2** должен вести себя, как объект типа A1 ***при любых значениях полей данных***.

Альтернативні варіанти ієрархії класів для зображення фігур на екрані



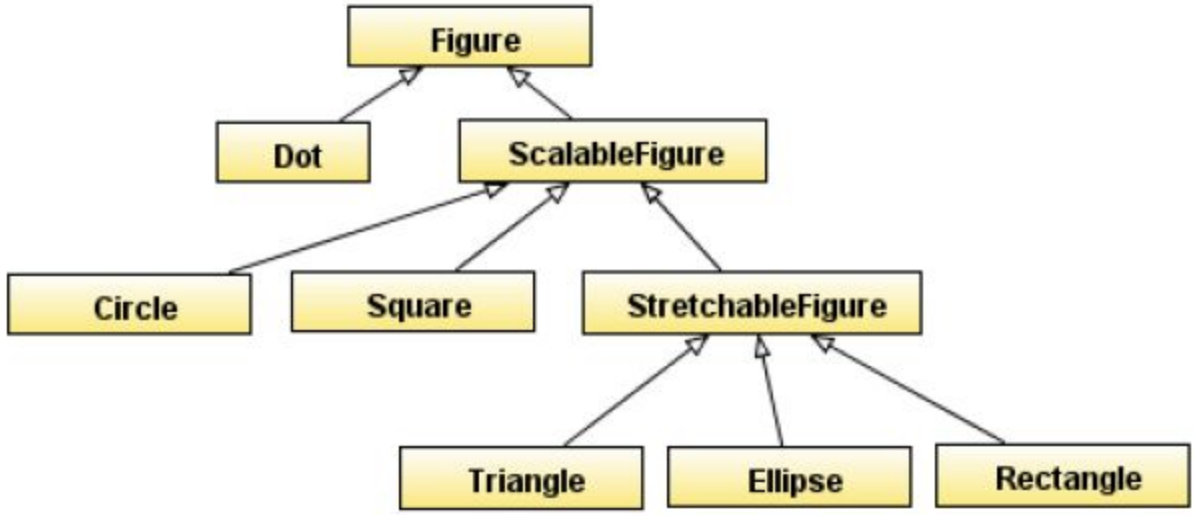


Один из важных принципов при построении иерархий — ***соответствие представлений из предметной области строящейся иерархии***.

Наследование от классов, у которых бывают экземпляры *(**неабстрактных**)*, нежелательно.

Классы, у которых нет экземпляров, называются ***абстрактными***.

Ієрархія з використанням абстрактних класів:



Основное преимущество такой иерархии по сравнению с предыдущими — возможность писать полиморфный код для наиболее общих разновидностей фигур.

Введение промежуточных уровней наследования, отвечающих соответствующим абстракциям, — характерная черта объектного программирования.
При этом классы **Figure**, **ScalableFigure** и **StretchableFigure** будут абстрактными — экземпляры такого типа создавать не предполагается, поскольку не бывает фигуры (масштабируемой или растягиваемой) в общем виде, без указания ее конкретной формы.

Продумывание того, как устроены классы, т. е. какие в них должны быть поля и методы (без уточнения конкретной реализации этих методов), а также описание того, какая должна быть иерархия наследования, называется ***проектированием***. Это сложный процесс, который гораздо важнее написания конкретных операторов в реализации *(**кодирования**)*.

Критерії правильності побудови ієрархії:

1. В процессе наследования должно идти расширение (усложнение, специализация, конкретизация) классов, а не наоборот.

2. Наследование должно идти только от абстрактных классов (или интерфейсов как варианта полностью абстрактных классов).

3. Названия всех методов должны давать точное представление о том, что делает метод. Причем эти имена должны восприниматься как команды (установить что-то, прочитать что-то, показать что-то и т. д.).

4. Недопустимы названия методов, содержащие связку "и" ("and"). Например, readAndShowSpeed, calculateIntegralAndWriteToFile и т. п. Такого рода гибриды следует разделять на два и более независимых метода — readSpeed и ShowSpeed; calculateIntegral и writeToFile, и т. д.

5. Названия всех классов должны давать четкое представление о соответствующих абстракциях поведения, в том числе для неабстрактных классов.

Спадкування. Перевизначення методів

При задании класса-потомка сначала идут модификаторы, затем после ключевого слова **class** идет имя декларируемого класса, потом идет зарезервированное слово **extends**, после чего требуется указать имя класса-родителя. Если не указывается, от какого класса идет наследование, то родителем считается класс **Object**.

Далее в фигурных скобках идет реализация класса — описание его полей и методов. При этом поля данных и методы, имеющиеся в прародителе, в потомке описывать не нужно — они наследуются. В классе-потомке прародительский метод можно реализовать по-другому. Тогда метод необходимо продекларировать и реализовать в классе-потомке. Кроме того, в потомке можно задавать новые поля данных и методы, отсутствующие в прародителях.

Модификаторы при задании класса-потомка:

- **public** — модификатор, задающий публичный (общедоступный) уровень видимости. Если он отсутствует, то действует пакетный уровень доступа — класс доступен только элементам того же пакета;
- **abstract** — модификатор, указывающий, что класс абстрактный, т. е. у него не бывает экземпляров (объектов). Обязательно объявлять класс абстрактным в случае, если какой-либо метод объявлен как абстрактный;
- **final** — модификатор, указывающий, что класс окончательный (final), т. е. что у него не может быть потомков.

Таким образом, задание класса-наследника имеет следующий формат:

```
Модификаторы class идентификатор extends суперкласс
{
        тело-класса
}
```

Пример: Задание абстрактного класса Figure

```
public abstract class Figure {  //абстрактный класс
int x=0;
int y=0;
java.awt.Color color;
java.awt.Graphics graphics;
java.awt.Color bgColor;
public abstract void show();    //абстрактный метод
public abstract void hide();    //абстрактный метод

public void moveTo(int x, int y){
hide();
this.x= x;
this.y= y;
show();
}
}
```

Пример: Задание класса Dot — наследника Figure

```
package java_gui_example;
import java.awt.*;
public class Dot extends Figure {  // Создает новый
                экземпляр типа Dot
public Dot(Graphics graphics,Color bgColor) {
this.graphics=graphics;
this.bgColor=bgColor; }
public void show() {
Color oldC=graphics.getColor();
graphics.setColor(Color.BLACK);
graphics.drawLine(x,y,x,y);
graphics.setColor(oldC); }
public void hide() {
Color oldC=graphics.getColor();
graphics.setColor(bgColor);
graphics.drawLine(x,y,x,y);
graphics.setColor(oldC); }
}
```

Спадкування і правила видимості

Поля и методы, помеченные как **private**, наследуются, но в классах-наследниках недоступны. Это сделано в целях обеспечения безопасности.

Модификатор **protected** предназначен для использования соответствующих полей и методов разработчиками классов-наследников. Он дает несколько большую открытость, чем пакетный вид доступа, поскольку в дополнение к видимости из текущего пакета позволяет обеспечить доступ к **protected** из классов-наследников, находящихся в других пакетах.

Модификатором **protected** полезно помечать различного рода служебные методы, ненужные пользователям класса, но необходимые для функциональности этого класса.

Зарезервоване слово super

Иногда возникает необходимость вызвать поле или метод из прародительского класса. Обычно это бывает в тех случаях, когда в классе-потомке задано поле с таким же именем или переопределен метод. В результате видимость прародительского поля данных или метода в классе-потомке утеряна. Говорят, что поле или метод ***затеняются*** в потомке.
В этих случаях используют вызов:

super.имяПоля или **super.имяМетода(список параметров)**

Слово **super** здесь означает сокращение от **superclass**. Если метод или поле заданы не в непосредственном прародителе, а унаследованы от более далекого прародителя, то соответствующие вызовы все равно будут работать, но комбинации вида **super.super.имя** не разрешены.
Вызовы с помощью слова **super** разрешены только для методов и полей данных объектов. Для методов и переменных класса вызовы с помощью ссылки **super** запрещены.

Клас Object

Классы Java являются узлами единого иерархического дерева. Корнем этого дерева является класс **Object** и, если определение суперкласса в объявлении какого-либо класса отсутствует, такой класс считается подклассом класса **Object** и в начале выполнения конструктора такого класса происходит обращение к конструктору класса **Object.**

В классе **Object** определен конструктор без параметров, который не выполняет никаких действий.

Методы класса **Object**:

protected Object clone() – создает и возвращает копию объекта;

protected void finalize() – вызывает «сборщик мусора», удаляющий объект, если на него больше нет ссылок;

public boolean equals(Object obj) – определяет равен ли один объект другому;

public final Class getClass() – получает объект типа **Class** для вызывающего объекта;

public int hashCode() – позволяет получить хэш-код объекта – целое число;

public String toString() возвращает строковое представление объекта – имя класса для данного объекта и хэш-код объекта ;

три перегруженных метода **wait(), notify()** и **notifyAll()** – используются для управления вычислительными потоками.

Перетворення змінних типу класів і масивів

В Java возможно преобразование переменных типа классов и массивов одного объектного типа в другой объектный тип. Как и для примитивных переменных, для ссылочных переменных определены расширяющие и сужающие преобразования.

Расширяющими преобразованиями для переменных типа классов и массивов являются **преобразования типа переменной подкласса в тип переменной суперкласса**, а также преобразование **null** в объект любого класса. Такие преобразования не требуют никаких действий во время выполнения и не приводят к ошибкам.

Сужающими преобразованиями для переменных типа классов и массивов являются **преобразования типа переменной суперкласса в тип переменной подкласса**.

Сужающие преобразования выполняются, как и для примитивных типов, с помощью оператора приведения типа:

(имя-класса)имя-переменной

Примеры преобразования типов:

```
class MyClass {…}
…
class MyFirstClass extends MyClass {…}
MyClass var1, var2;
MyFirstClass fvar1, fvar1, fvar3;
Object objvar, objvar3;
…
objvar = fvar;      // Расширяющее преобразование
var1 = fvar1;             // Расширяющее преобразование
fvar2 = (MyFirstClass)var2;       // Сужающее преобразование
fvar3 = (MyFirstClass)objvar3;  // Сужающее преобразование
```