и **r**). Каким образом метод **printFeatures()** может обрабатывать объекты двух различных классов? Все дело в типе передаваемого этому методу аргумента – класса, реализующего интерфейс **Shape**. Вызывать, однако, можно только те методы, которые были объявлены в интерфейсе.

В следующем примере в классе **ShapeCreator** используются классы и интерфейсы, определенные выше, и объявляется ссылка на интерфейсный тип. Такая ссылка может указывать на экземпляр любого класса, который реализует объявленный интерфейс. При вызове метода через такую ссылку будет вызываться его реализованная версия, основанная на текущем экземпляре класса. Выполняемый метод разыскивается динамически во время выполнения, что позволяет создавать классы позже кода, который вызывает их методы.

```
/* пример # 6 : динамический связывание методов : ShapeCreator.java */
package chapt06;

public class ShapeCreator {
      public static void main(String[] args) {
            Shape sh; /* ссылка на интерфейсный тип */
            Rectangle re = new Rectangle(5, 9.95);
            sh = re;
            sh.getPerimeter();//вызов метода класса Rectangle
            Circle cr = new Circle(7.01);
            sh = cr; // присваивается ссылка на другой объект
            sh.getPerimeter();//вызов метода класса Circle

            // cr=re; // ошибка! разные ветви наследования
      }
}
```

Невозможно приравнивать ссылки на классы, находящиеся в разных ветвях наследования, так как не существует никакого способа привести один такой тип к другому. По этой же причине ошибку вызовет попытка объявления объекта в виде:

```
Circle c = new Rectangle(1, 5);
```

## Пакеты

Любой класс Java относится к определенному пакету, который может быть неименованным (unnamed или default package), если оператор **package** отсутствует. Оператор **package** имя, помещаемый в начале исходного программного файла, определяет именованный пакет, т.е. область в пространстве имен классов, где определяются имена классов, содержащихся в этом файле. Действие оператора **package** указывает на месторасположение файла относительно корневого каталога проекта. Например:

```
package chapt06;
```

При этом программный файл будет помещен в подкаталог с названием **chapt06**. Имя пакета при обращении к классу из другого пакета присоединяется к имени класса: **chapt06.Student**. Внутри указанной области можно выделить подобласти:

```
package chapt06.bsu;
```

Общая форма файла, содержащего исходный код Java, может быть следующая:

```
одиночный оператор package (необязателен);
любое количество операторов import (необязательны);
одиночный открытый (public) класс (необязателен)
любое количество классов пакета (необязательны)
```

В реальных проектах пакеты часто именуются следующим образом:

- обратный интернет-адрес производителя программного обеспечения, а именно для `www.bsu.by` получится `by.bsu`;
- далее следует имя проекта, например: `eun`;
- затем располагаются пакеты, определяющие собственно приложение.

При использовании классов перед именем класса через точку надо добавлять полное имя пакета, к которому относится данный класс. На рисунке приведен далеко не полный список пакетов реального приложения. Из названий пакетов можно определить, какие примерно классы в нем расположены, не заглядывая внутрь. При создании пакета всегда следует руководствоваться простым правилом: называть его именем простым, но отражающим смысл, логику поведения и функциональность объединенных в нем классов.

by.bsu.eun
by.bsu.eun.administration.constants
by.bsu.eun.administration.dbhelpers
by.bsu.eun.common.constants
by.bsu.eun.common.dbhelpers.annboard
by.bsu.eun.common.dbhelpers.courses
by.bsu.eun.common.dbhelpers.guestbook
by.bsu.eun.common.dbhelpers.learnres
by.bsu.eun.common.dbhelpers.messages
by.bsu.eun.common.dbhelpers.news
by.bsu.eun.common.dbhelpers.prepinfo
by.bsu.eun.common.dbhelpers.statistics
by.bsu.eun.common.dbhelpers.subjectmark
by.bsu.eun.common.dbhelpers.subjects
by.bsu.eun.common.dbhelpers.test
by.bsu.eun.common.dbhelpers.users
by.bsu.eun.common.menus
by.bsu.eun.common.objects
by.bsu.eun.common.servlets
by.bsu.eun.common.tools
by.bsu.eun.consultation.constants
by.bsu.eun.consultation.dbhelpers
by.bsu.eun.consultation.objects
by.bsu.eun.core.constants
by.bsu.eun.core.dbhelpers
by.bsu.eun.core.exceptions
by.bsu.eun.core.filters
by.bsu.eun.core.managers
by.bsu.eun.core.taglibs

**Рис. 6.1.** Организация пакетов приложения

Каждый класс добавляется в указанный пакет при компиляции. Например:

```
// пример # 7 : простейший класс в пакете: CommonObject.java
package by.bsu.eun.objects;

public class CommonObject implements Cloneable {
      public CommonObject() {
            super();
      }
      public Object clone()
            throws CloneNotSupportedException {
            return super.clone();
      }
}
```

Класс начинается с указания того, что он принадлежит пакету **by.bsu.eun.objects**. Другими словами, это означает, что файл **CommonObject.java** находится в каталоге **objects**, который, в свою очередь, находится в каталоге **bsu**, и так далее. Нельзя переименовывать пакет, не переименовав каталог, в котором хранятся его классы. Чтобы получить доступ к классу из другого пакета, перед именем такого класса указывается имя пакета: **by.bsu.eun.objects.CommonObject**. Чтобы избежать таких длинных имен, используется ключевое слово **import.** Например:

```
import by.bsu.eun.objects.CommonObject;
```

или

```
import by.bsu.eun.objects.*;
```

Во втором варианте импортируется весь пакет, что означает возможность доступа к любому классу пакета, но только не к подпакету и его классам. В практическом программировании следует использовать индивидуальный **import** класса, чтобы при анализе кода была возможность быстро определить месторасположение используемого класса.

Доступ к классу из другого пакета можно осуществить следующим образом:

```
// пример # 8 : доступ к пакету: UserStatistic.java
package by.bsu.eun.usermng;

public class UserStatistic
                  extends by.bsu.eun.objects.CommonObject {
      private long id;
      private int mark ;

      public UserStatistic() {
            super();
      }
      public long getId() {
            return id;
      }
      public void setId(long id) {
```

```
            this.id = id;
      }
      public int getMark() {
            return mark;
      }
      public void setMark(int mark) {
            this.mark = mark;
      }
}
```

При импорте класса из другого пакета рекомендуется всегда указывать полный путь с указанием имени импортируемого класса. Это позволяет в большом проекте легко найти определение класса, если возникает необходимость посмотреть исходный код класса.

```
// пример # 9 : доступ к пакету: CreatorStatistic.java
package by.bsu.eun.actions;
import by.bsu.eun.objects.CommonObject;
import by.bsu.eun.usermng.UserStatistic;

public class CreatorStatistic {
   public static UserStatistic createUserStatistic(long id)
{
            UserStatistic temp = new UserStatistic();
            temp.setId(id);
            // чтение информации из базы данных по id пользователя
            int mark = полученное значение;
            temp.setMark(mark);
            return temp;
      }
      public static void main(String[] args) {
            UserStatistic us = createUserStatistic(71);
            System.out.println(us.getMark());
      }
}
```

Если пакет не существует, то его необходимо создать до первой компиляции, если пакет не указан, класс добавляется в пакет без имени (unnamed). При этом unnamed-каталог не создается. Однако в реальных проектах классы вне пакетов не создаются, и не существует причин отступать от этого правила.

### Статический импорт

Константы и статические методы класса можно использовать без указания принадлежности к классу, если применить статический импорт, как это показано в следующем примере.

```
// пример # 10 : статический импорт: ImportDemo.java
package chapt06;
import static java.lang.Math.*;

public class ImportDemo {
```