Тестовые задания к главе 1

Вопрос 1.1.

Дан код программ:

A)

```
public class Quest21 {
   public static void main (String [] args) {
   System.out.println ("Hello, java 7");
   }
}
```

B)

```
public class Quest22 {
   String java = "Java 7";
   public static void main (String [] args) {
      System.out.println (java);
   }
}
```

C)

```
public class Quest23 {
{
   System.out.println ("Java 7");
}
}
```

Укажите, что скомпилируется без ошибок (выберите 1).

1) AB
2) BC
3) ABC
4) A
5) AC

Вопрос 1.2.

Выберите правильные утверждения (2):

1) класс — это тип данных;
2) объект класса может использоваться всюду, где используется объект подкласса;
3) объект класса можно создать только один раз;
4) на объект класса может не ссылаться объектная переменная.

Вопрос 1.3.

Вставьте на место прочерка название одного из принципов ООП так, чтобы получилось верное определение (1):

1) наследование
2) полиморфизм
3) позднее связывание
4) инкапсуляция

_____ *— это объединение данных и методов, предназначенных для манипулирования этими данными в новом типе — классе.*

Вопрос 1.4.

Дан код:

```
String s; // 1
if ((s = "java") == "java") {// 2
System.out.println (s+ " true");
} else {
System.out.println (s+ " false");
}
```

Что будет результатом компиляции и запуска этого кода (1)?

1) ошибка компиляции в строке 1, переменная не проинициализирована
2) ошибка компиляции в строке 2, неправильное выражение для оператора if
3) на консоль выведется **java true**
4) на консоль выведется **java false**

Вопрос 1.5.
Дан код:
```
public class Quest5 {
   private static void main (String [] args) {
      System.out.println (args [2]);
   }
}
```
Что произойдет, если этот код выполняется следующей командной строкой:
java Quest5 Java 7"" (1)?
1) выведется: Java 7
2) ошибка времени выполнения NullPointerException
3) ошибка времени выполнения ArraylndexOutOfBoundsException
4) выведется: Java 7 <пустая строка>
5) выведется: <пустая строка>
6) приложение не запустится
7) код не скомпилируется

Вопрос 1.6.
Дан код:
```
public class Quest6 {
public static void main (String [] args) {
   System.out.print ("A");
   main ("java7");
}
private static void main (String args) {
   System.out.print ("B");
}
}
```
Что будет выведено в результате запуска и компиляции (1)?
1) ошибка компиляции
2) BA
3) AB
4) AA
5) компиляция пройдет успешно, а при выполнении программа зациклится

Вопрос 1.7.
Дан код:
```
class Book {
   private String book;
   public void setBook (String b) {book = b;}
}
public class Quest7 {
   public static void main (String [] args) {
      Book book1 = new Book (); book1.setBook ("Java 7");
      Book book2 = new Book (); book2.setBook ("Java 7");
      if (book1.equals (book2)) {
         System.out.println ("True");
      } else {
         System.out.println ("False");
      }
   }
}
```
Результатом компиляции и запуска кода будет (1)?
1) True
2) ошибка компиляции
3) False

Ответы к главе 1

Вопрос 1.1.

Код класса **Quest21** ошибок не содержит; статический метод **main()** класса **Quest22** пытается получить доступ к нестатическому полю, описанному в этом классе, что является ошибкой; код класса **Quest23** включает логический блок инициализации, внутри которого вызов методов допустим, код данного класса скомпилируется без ошибок, однако при запуске возникнет исключительная ситуация типа **java.lang.NoClassSuchMethodError**.

Ответ: 5.

Вопрос 1.2.

Класс — это тип данных; если он разработан программистом-пользователем системы, то говорят, что класс — это пользовательский тип данных. На объект класса действительно может не ссылаться объектная переменная, если при создании объекта этого явно не указать, отработать такой объект может только один раз в точке своего создания. Объект класса может использоваться всюду, где используется объект ЕГО (а не любого) подкласса; так, имея отношение наследования *Человек -> Доктор* получаем, что доктора (объект производного класса) можно отправить за покупками в магазин так же, как и человека (объект базового класса). Если конструкцией класса не определено, то объектов класса можно создать сколько угодно, пока есть свободная память.

Ответ:1, 4.

Вопрос 1.3.

Наследование — приобретение свойств другого объекта или класса. Полиморфизм позволяет каждому типу объекта определять свое собственное поведение. Связыванием называется сопоставление вызова функции с телом функции. Позднее связывание производится во время выполнения программы в зависимости от фактического типа объекта. Инкапсуляция — это механизм, позволяющий объединить в классе данные и методы, работающие с ними, и скрыть детали реализации.

Ответ: 4.

Вопрос 1.4.

Ошибок компиляции данный фрагмент кода не вызовет. Ссылка **s** инициализируется перед первым использованием. Сравнение в операторе **if** производится по ссылкам на строку, определенную в пуле литералов.

Ответ: 3.

Вопрос 1.5.

Данное приложение не запустится, так как статический метод **main** имеет атрибут доступа **private**, который не позволяет вызвать данный метод на выполнение вне класса **Quest5**. После успешной компиляции и запуска с помощью командной строки **Quest6 Java 7** "" появится сообщение **Error: Main method not found in class Quest5, please define the main method as: public static void main(String[] args)**.

Ответ: 6.

Вопрос 1.6.

При запуске программы вызовется метод **main** с атрибутом доступа **public**, принимающий в качестве параметра массив строк. В теле этого метода вызывается перегруженный метод **main**, принимающий в качестве параметра только один строковый аргумент.

Ответ: 3.

Вопрос 1.7.

Класс **Book** не имеет собственной реализации метода **equals**, данный метод (возвращающий **true** только при тождественности сравниваемых ссылок) наследуется от класса **Object**.

Ответ: 3.

Ответы:

1.1. — 5 1.4. — 3 1.6. — 3
1.2. — 1, 4 1.5. — 6 1.7. — 3
1.3. — 4

Тестовые задания к главе 2

Вопрос 2.1.

Укажите строки, компиляция которых не приведет к ошибке (3):

1) **int** var1 = 356f
2) **double** var2 = 356f
3) **float** var3 = 356f
4) **char** var4 = 356f
5) **long** var5 = 356f
6) **byte** var6 = 356f
7) Integer var7 = 356f
8) Character var8 = 356f
9) Object var9 = 356f

Вопрос 2.2.

Укажите, какие javadoc-комментарии не используются для документирования конструкторов и методов (2):

1) @see
2) @author
3) @param
4) @version
5) @throws
6) @deprecated

Вопрос 2.3.

Дан код:

```
public class Quest4 {
   public static void main (String [] args) {
      double x=0, y=2, z;
      z = y/x;
      System.out.println ("z="+z);
   }
}
```

Что выведется на консоль в результате компиляции и запуска программы (1)?

1) Ошибка компиляции
2) z=Infinity
3) z=NaN
4) Ошибка времени выполнения java.lang.ArithmeticException

Вопрос 2.4.

Что будет результатом компиляции и запуска следующего кода (1)?

```
public class Quest {
   public static void main (String [] args) {
      MedicalStaff medic = new HeadDoctor ();
      if (medic instanceof Nurse) {
         System.out.println ("Nurse");
      } else if (medic instanceof Doctor) {
         System.out.println ("Doctor");
      } else if (medic instanceof HeadDoctor) {
         System.out.println ("HeadDoctor");
      }
   }
}
class MedicalStaff {}
class Doctor extends MedicalStaff {}
class Nurse extends MedicalStaff {}
class HeadDoctor extends Doctor {}
```

1) Nurse
2) Doctor
3) HeadDoctor
4) Ошибка компиляции

Вопрос 2.5.
Дан фрагмент кода **if** (e1) **if** (e2) S1; **else** S2; (e1, e2, S1, S2 — корректные java-выражения). Какому другому фрагменту кода он эквивалентен (2)?
1) **if** (e1) {**if** (e2) S1; **else** S2;}
2) **if** (e1) {**if** (e2) S1;} **else** S2;
3) **if** (e1) **if** (e2) S1; **else**; **else** S2;
4) **if** (e1) **if** (e2) S1; **else** S2; **else**;

Вопрос 2.6.
Какие из фрагментов кода неверно решают задачу «Найти сумму первых 100 натуральных чисел» (2)?
1) i = 1; sum = 0; **for** (; i <= 100; i++) sum += i;
2) sum = 0; **for** (i = 1; i <= 100;) sum += i++;
3) **for** (i = 1, sum = 0; i <= 100; sum += i+, i++);
4) **for** (i = 1, sum = 0; i <= 100; sum += i++);
5) **for** (i = 0, sum = 0; i++, i <= 100; sum += i);

Вопрос 2.7.
Какие утверждения о классах-оболочках корректны (3)?
1) Классы оболочки Double, Long, Float размещаются в пакете java.util
2) Объекты классов оболочек могут хранить те же значения, что и соответствующие им базовые типы
3) Объекты классов-оболочек хранят изменяемые значения аналогично переменным базовых типов
4) Объекты классов-оболочек по умолчанию получают значение null
5) В классах оболочках определены методы преобразования к базовому типу

Вопрос 2.8.
Дан код:
class Item {}
1) **int** [] mas1 = **new int** [24];
2) Integer mas2 [] = **new** Integer [24];
3) **char** [] mas3 = **new** Character [] {'a', 'b', 'c'};
4) Item [] mas4 = **new** Item {**new** Item (), **new** Item ()};
5) **double** [] mas5 = {5, 10, 15, 20};
6) **int** [] mas6 [] = **new int** [4] [5];
7) **int** mas7 [] [] = **new int** [4] [];
Компиляция каких строк приведет к ошибке (2)?

Ответы к главе 2

Вопрос 2.1.
Константа **356f** — константа типа **float**. Неявное преобразование константы типа **float** возможно к типу **double, float, Object (Float)**.
Ответ: 2, 3, 9.

Вопрос 2.2.
Теги **@author** и **@version** не используются для описания конструкторов и методов, их применяют для описания пакетов и классов.
Ответ: 2, 4.

Вопрос 2.3.
При делении на вещественный нуль приводит к возбуждению исключительной ситуации. Результатом, в зависимости от делимого, будет **+/- Infinity**.
Ответ: 2.

Вопрос 2.4.
Объект, на который ссылается объектная переменная **medical**, является объектом класса **HeadDoctor**, производным от класса **Doctor**. Объект типа **HeadDoctor** является экземпляром типа **Doctor.**
Ответ: 2.

Вопрос 2.5.
Искомому фрагменту кода и фрагментам под номерами 1 и 4 соответствует блок схема:
А фрагменты под номерами 2 и 3 отвечают следующей блок-схеме:
Ответ: 1, 4.

Вопрос 2.6.
В циклах под номерами 3 и 5 допущены синтаксические ошибки. Циклы под номерами 1, 3 и 4, вычисляют сумму первых ста натуральных чисел.
Ответ: 3, 5.

Вопрос 2.7.
Классы оболочки расположены в пакете **java.lang**. Объекты классов оболочек хранят неизменяемые значения.
Ответ: 2, 4, 5.

Вопрос 2.8.
Компиляция строки 3 приведет к ошибке **Type mismatch: cannot convert from Character[] to char[].** Строка 4 содержит синтаксическую ошибку — отсутствуют квадратные скобки, указывающие, что создается массив.
Ответ: 3, 4.

Ответы:

2.1. — 2, 3, 9	2.4. — 2	2.7. — 2, 4, 5
2.2. — 2, 4	2.5. — 1, 4	2.8. — 3, 4
2.3. — 2	2.6. — 3, 5	

Тестовые задания к главе 3

Вопрос 3.1.
Какие описания класса содержат синтаксическую ошибку? Код написан в файле Quest1.java (3)
1) **public class** Quest1 {}
2) **public static class** Quest1 {}
3) **public abstract final class** Quest1 {}
4) **private class** Quest1 {}
5) **final class** Quest1 {}

Вопрос 3.2.
Выберите правильное утверждение, подходящее для окончания фразы «Константное поле может быть проинициализировано …» (3):
1) только один раз;
2) один раз при объявлении, а затем в конструкторе класса;
3) в логическом блоке инициализации;
4) в статическом блоке инициализации;
5) при объявлении или в конструкторе класса.

Вопрос 3.3.
Дан код:

```
public class Quest3 {
   public static int method () {
      final int loc;
      System.out.println (loc);//1
      loc=4;//2
      return loc+1;//3
   }
   public static void main (String [] args) {
      method (); method (); method ();
      System.out.println (method ());
   }
}
```

Каким будет результат компиляции и запуска программы (1)?
1) на консоль выведется число 4
2) на консоль выведется число 0
3) ошибка компиляции в строке 1
4) ошибка компиляции в строке 2
5) ошибка компиляции в строке 3

Вопрос 3.4.
Выберите утверждения, корректно характеризующие модификаторы доступа (2):
1) статические private-члены класса доступны только статическим методам этого класса;
2) статические public-члены классы доступны всем методам этого класса;
3) protected-члены класса доступны подклассам другого пакета;
4) поле — член класса, объявленное без модификатора доступа, доступно в классах другого пакета.

Вопрос 3.5.
Дан код:

```
public class Quest5 {
   public Quest5 () {}
   public Quest5 (int i) {this (i, i);}
   public Quest5 (int i, int j) {this ();}
   public static void main (String [] args) {
      Quest5 q = new Quest5 (2,3); //1
   }
}
```

Сколько конструкторов вызовется при создании объекта в строке 1 (1)?
1) один;
2) два;
3) три;
4) ошибка компиляции.

Вопрос 3.6.
Дан код (1):
```
public class Quest6 {
   public void meth (Number obj) {System.out.print ("1");}
   public void meth (Character obj) {System.out.print ("2");}
   private static void meth (Integer obj) {System.out.print ("3");}
   public void meth (int i) {System.out.print ("4");}
   public void meth (double d) {System.out.print ("5");}
   public static void main (String [] args) {
      Quest6 q = new Quest6 ();
      Number n = 67;
      Integer i = 78;
      q.meth (n);
      q.meth (i);
   }
}
```
Что выведется на консоль после компиляции и запуска этой программы?
1) 14
2) **11**
3) **33**
4) 44
5) **13**
6) ошибка компиляции
7) ошибка выполнения

Вопрос 3.7.
Дан код**:**
```
public class Quest7<T> {
   private T pole;
   public Quest7 (T pole) {this.pole = pole;} //1
   public void setPoleDefault () {pole.setTime (1000);} //2
   public static void main (String [] args) {
      Quest7<Date> obj = new Quest7<Date> (new Date ());//3
      obj.setPoleDefault ();
   }
}
```
Каким будет результат компиляции и запуска программы (1)?
1) компиляция и запуск пройдут без ошибок
2) ошибка компиляции в строке 1
3) ошибка компиляции в строке 2
4) ошибка компиляции в строке 3

Вопрос 3.8.
Укажите корректый способ создания экземпляра класса
`public class Quest8<T1, T2> {}`? (2)
ОСНОВЫ JAVA
1) `Quest8 obj = new Quest8 ()`
2) `Quest8<Object> obj = new Quest8<Object> ()`
3) `Quest8<Object, Object> obj = new Quest8<Object, Object> ()`
4) `Quest8<…, Object> obj = new Quest8<…, Object> ()`
5) `Quest8<Object, Integer> obj = new Quest8<Integer, Object> ()`
6) `Quest8<Number, Integer> obj = new Quest8<Integer, Integer> ()`

Вопрос 3.9.
Дан код:

```
enum Numbers {ONE, TWO, THREE, FOUR, FIVE}
public class Quest9 {
   public static void main (String [] args) {
      Numbers n1 = Numbers.ONE;
      Numbers n2 = Numbers.ONE;//1
      if (n1 == n2) {System.out.print ("true");}
      else {System.out.print ("false");}
      System.out.println (Numbers.FIVE.ordinal ());//2
   }
}
```

Что выведется на консоль в результате компиляции и запуска приложения (1)?
1) false4
2) true4
3) false5
4) true5
5) произойдет ошибка компиляции в строке 1
6) произойдет ошибка компиляции в строке 2

Вопрос 3.10.
Выберите неправильные утверждения (3):
1) перечисление является классом
2) при объявлении перечисления его необходимо явно наследовать от класса **java.lang.Enum**
3) в теле перечисления можно объявлять только методы
4) конструктор перечисления может быть объявлен со спецификатором **p**

Ответы к главе 3

Вопрос 3.1.
Корректными являются описания классов 1) и 5). При описании класса **Quest1** в файле **Quest1.java** допустимо использовать модификаторы **public**, **abstract** или **final**.
Ответ: 2, 3, 4.

Вопрос 3.2.
Константное поле экземпляра класса может быть проинициализировано только один раз либо при объявлении, либо в логическом блоке инициализации, либо в конструкторе класса.
Ответ: 1, 3, 5.

Вопрос 3.3.
При компиляции данной программы будет ошибка в строке 1, так как локальная переменная используется без инициализации.
Ответ: 3.

Вопрос 3.4.
Статические **private**-поля класса доступны во всех методах этого класса, как и статические **public**-поля. К **protected**-полям класса есть доступ из подклассов, находящихся в другом пакете. Поля класса, объявленные без модификатора доступа (friendly), видны в классах этого же пакета.
Ответ: 2, 3.

Вопрос 3.5.
При создании объекта класса **Quest5** в строке 1 вызовется 3 конструктора, один класса **Object**, и два класса **Quest5.**
Ответ: 3.

Вопрос 3.6.
Для разрешения перегрузки сначала ищется метод, тип формального параметра которого совпадает с типом фактического параметра, и только в случае неудачи ищется другой метод, к типу формального параметра которого можно преобразовать передаваемый объект.
Ответ: 5.

Вопрос 3.7.
Параметризованным типам разрешено вызывать только методы класса **Object.**
Ответ: 3.

Вопрос 3.8.
Второй вариант создания экземпляра параметризованного класса содержит неверное число аргументов, четвертый — объявлен с неправильным синтаксисом, пятый и шестой содержат ошибку преобразования типа.
Ответ: 1, 3.

Вопрос 3.9.
Объекты перечисления создаются в единственном экземпляре, поэтому сравнение оператором «==» элементов перечисления приводит к сравнению их объектных ссылок. Нумерация позиций элементов перечисления начинается с нуля.
Ответ: 2.

Вопрос 3.10.
В теле перечисления можно объявлять методы, поля и конструкторы. Конструктор перечисления определяется без модификатора доступа или с модификатором **private.**
Перечисления неявно наследуются от класса **java.lang.Enum**.
Ответ: 2,3,4.

Ответы:

3.1. — 2, 3, 4	3.3. — 3	3.5. — 3	3.7. — 3	3.9. — 2
3.2. — 1, 3, 5	3.4. — 2, 3	3.6. — 5	3.8. — 1, 3	3.10. — 2, 3, 4

`

Тестовые задания к главе 4

Вопрос 4.1.
Дан класс**:**

```
package ch04.q01;
class Quest41 {}
```

Укажите правильные варианты наследования от этого класса (2):

```
1) package ch04.q01; class Quest4 extends Quest41 {}
2) package ch04.q01._2; public class Quest42 extends Quest41 {}
3) package ch04.q01; public class Quest43 implements Quest41 {}
4) package ch04.q01._2; import ch04.q01.Quest41;
5) public class Quest44 extends Quest41 {}
6) package ch04.q01; public class Quest45 extends Quest41 {}
```

Вопрос 4.2.
Выберите правильные утверждения (3):
1) Класс может быть использован в качестве суперкласса для себя самого.
2) В конструкторе класса можно совместно использовать вызовы this и super.
3) Статические методы можно определять в подклассах с той же сигнатурой, что и в базовом классе.
4) Статические методы можно перегружать в подклассах.
5) Динамическое связывание определяет версию вызываемого метода на этапе выполнения.

Вопрос 4.3.
Дан код:
```
package ch04.q03;
public class Quest43 {
   private final void method () {} //1
}
class Quest431 extends Quest43 {
   public void method () {} //2
}
```
Что произойдет в результате компиляции этого кода (1)?
1) ошибка компиляции в строке 1
2) ошибка компиляции в строке 2
3) компиляция без ошибок

Вопрос 4.4.
Дан код двух классов**:**
```
// класс 1
package ch04.q04;
public class Quest41 {}
// класс 2
package ch04.q04._2;
import ch04.q04.Quest41;
public class Quest43 extends Quest41 {
   public Quest43 () {
      super ();
   }
}
```
С каким атрибутом доступа объявлен конструктор по умолчанию в базовом классе Quest41 (1)?
1) public
2) private
3) protected
4) friendly

Вопрос 4.5.
Дан код:
```
package ch04.q05;
public class Quest51 {
   public String toString () {
   return getClass ().getSimpleName ();
   }
   public static void main (String [] args) {
      Quest53 q = new Quest53 ();
      System.out.println (q.toString ());
   }
}
class Quest52 extends Quest51 {}
class Quest53 extends Quest52 {}
```
Что выведется на консоль в результате компиляции и запуска программы (1)?
1) Quest52
2) Quest53
3) Quest51
4) ошибка компиляции

Вопрос 4.6.
Дан код**:**

```
package ch04.q06;
class Item {
   public int item;
   Item (int item) {
      this.item = item;
   }
}
public class Quest61 {
   public static void main (String [] args) {
      Item ar1 [] = {new Item (1), new Item (2), new Item (3)};
      Item ar2 [] = ar1.clone ();
      ar2 [0].item = 4;
      System.out.println (ar1 [0].item +" " + ar1 [1].item + " " + ar1 [2].item);
   }
}
```

Что выведется на консоль после компиляции и запуска этой программы (1)?
1) 1 2 3
2) 1 4 3
3) 4 2 3
4) ошибка компиляции
5) ошибка выполнения

Ответы к главе 4

Вопрос 4.1.
Класс **Quest41** объявлен в пакете с атрибутом friendly, значит, доступен для наследования классам этого пакета. Для наследования применяется ключевое слово **extends**,
а не **implements**.
Ответ: 1, 6.

Вопрос 4.2.
Класс не может наследоваться от самого себя. В конструкторе класса нельзя совместно использовать вызовы **super()** и **this()**, поскольку такой вызов должен быть всегда первым оператором конструктора. Компилятор не помешает переопределить статические методы в подклассах, однако при их вызове будет использоваться механизм раннего связывания. Аннотацию **@Override** к статическим методам применять нельзя. Статические методы можно перегружать в подклассах, доступность таких методов зависит от типа ссылки и атрибута доступа. При динамическом связывании версия вызываемого метода определяется на этапе выполнения.
Ответ: 3, 4, 5.

Вопрос 4.3.
Компиляция этого кода завершится без ошибок, закрытый **final-**метод класса **Quest43** не переопределяется открытым методом класса **Quest431**.
Ответ: 3.

Вопрос 4.4.
Если программист не определил ни одного конструктора, компилятор добавляет в класс конструктор по умолчанию с атрибутом доступа **public**.
Ответ: 1.

Вопрос 4.5.
При вызове **getClass().getSimpleName()** в виде строки вернется имя класса, от которого был создан объект.
Ответ: 2.

Вопрос 4.6.
Для массивов Java переопределен метод **clone()**, который производит поэлементное копирование.
Ответ: 3.

Ответы:

4.1. — 1,6	4.3. — 3	4.5. — 2
4.2. — 3,4,5	4.4. — 1	4.6. — 3

Тестовые задания к главе 5

Вопрос 5.1.
Дан код**:**

```
public class Quest1 {
   public static void main (String [] args) {
      for (Numbers num: Numbers.values ()) {
         System.out.print (num.getNumber ());
      }
   }
}
enum Numbers {
   ONE (1),
   TWO (2) {public int getNumber () {return x + x;}},
   THREE (3) {public int getNumber () {return x + x + x;}},
   FOUR (4), FIVE (5);
   int x;
   Numbers (int x) {
      this.x = x;
   }
   public int getNumber () {
      return x;
   }
}
```

Что будет результатом компиляции и выполнения данного кода (1)?
1) строка 12345
2) строка 54321
3) строка 14945
4) строка 54941
5) строка 12945
6) строка 54921
7) строка 14345
8) строка 54341
9) ошибка компиляции

Вопрос 5.2.
Дан код:
```
public class Quest3 {
   public static void main (String [] args) {
      Outer obj = new Outer ().new Inner1 ();
      obj.print ();
   }
}
class Outer {
   public void print () {}
   class Inner1 _____ // line 1{
      public void print () {
      System.out.println ("In inner.");
      }
   }
}
```
Что необходимо дописать в line 1, чтобы при компиляции и запуске на консоль вывелась строка In inner (1)?
1) ничего, код написан верно
2) **implements** Outer
3) **extends** Outer
4) нет верного варианта

Вопрос 5.3.
Выберите правильный вариант доступа из внутреннего Inner класса к экземпляру его внешнего Outer класса (1):
1) Outer.**class.this;**
2) **new** Outer ().**this;**
3) Inner.Outer.**class.this;**
4) Outer.**class**.newInstance ().**this;**
5) Outer.**this.**

Вопрос 5.4.
Даны два фрагмента кода:
Фрагмент 1
```
new Object () {
   public void hello () {
      System.out.print ("Hello!");
   }
}.hello ();
```

Фрагмент 2
```
Object obj = new Object () {
   public void hello () {
      System.out.print ("Hello!");
   }
};
obj.hello ();
```
Каким будет результат компиляции и запуска этих фрагментов кода (1)?
1) и первый, и второй фрагменты кода скомпилируются и выведут на консоль строку «Hello!»;
2) первый фрагмент кода скомпилируется и выведет на консоль строку «Hello!», при компиляции второго фрагмента возникнет ошибка;
3) второй фрагмент кода скомпилируется и выведет на консоль строку «Hello!», при компиляции первого фрагмента возникнет ошибка;
4) ни первый, ни второй фрагмент кода не скомпилируются.

Вопрос 5.5.
Выберите неправильные утверждения (5):
1) методы внутреннего (нестатического) класса имеют прямой доступ только к статическим полям и методам внешнего класса;
2) доступ к элементам внутреннего класса возможен только из внешнего класса через объект внутреннего класса;
3) внутренние классы могут содержать static-поля;
4) внутренние классы не могут реализовывать интерфейсы;
5) внутренние классы могут быть объявлены только с параметрами final, abstract, public, protected, private;
6) внутренний локальный класс обладает доступом только к полям метода, в котором он объявлен;
7) внутренние классы могут содержать static-поля, только если они final static;
8) внутренние классы могут быть производными классами, но не базовыми

Ответы к главе 5

Вопрос 5.1.
Новая версия метода **getNumber()** будет вызываться для элементов перечисления ***TWO*** *и* ***THREE*****.**
Ответ: 3.

Вопрос 5.2.
Внутренние классы могут быть производными, а ссылка базового типа может ссылаться на объект производного.
Ответ: 3.

Вопрос 5.3.
Для получения доступа из внутреннего класса к экземпляру его внешнего класса необходимо в ссылке указать имя класса и ключевое слово **this**, поставив между ними точку (например, **Outer.this**).
Ответ: 5.

Вопрос 5.4.
Во втором фрагменте кода создается объект от анонимного класса, на объект указывает ссылка типа **Object**, для которого метод **hello()** не определен.
Ответ: 2.

Вопрос 5.5.
Методы внутреннего класса имеют прямой доступ ко всем полям и методам внешнего класса. Внутренние классы не могут содержать **static**-полей, кроме **final static**.
Внутренние классы могут реализовывать интерфейсы. Внутренний класс может быть объявлен внутри метода или логического блока внешнего класса; видимость класса регулируется видимостью того блока, в котором он объявлен; однако класс сохраняет доступ ко всем полям и методам внешнего класса, а также константам, объявленным в текущем блоке кода. Внутренние классы могут быть как производными, так и базовыми классами.
Ответ: 1, 3, 4, 6, 8.

Ответы:

5.1. — 3	5.3. — 5	5.5. — 1,3,4,6,8
5.2. — 4	5.4. — 2	

Тестовые задания к главе 6

Вопрос 6.1.

Что будет результатом компиляции и запуска следующего кода (1)?

```
interface Quest10{ Number returner(); }
abstract class Quest100{
   public abstract Integer returner();
}
public class Quest1 extends Quest100 implements Quest10{//line 1
   @Override //line 2
   public Integer returner() {// line 3
      return new Integer(6);
   }
   public static void main(String[] args) {
      Quest1 quest = new Quest1();
      Quest10 quest10 = quest;
      Quest100 quest100 = quest;
      System.out.println(quest10.returner()+""+quest100.returner());
   }
}
```

1) ошибка компиляции в строке 1
2) ошибка компиляции в строке 2
3) ошибка компиляции в строке 3
4) компиляция и запуск программы осуществится без ошибок

Вопрос 6.2.

Даны объявления интерфейсов. Которые из них скомпилируются с ошибкой (2)?

1)
```
interface Quest20{
   class Inner{
      private int x;
   }
}
```
2)
```
interface Quest21{
   static class Inner{ int x; }
}
```
3)
```
class Quest22{
   interface Inner{ int x; }
}
```
4)
```
class Quest23{
   static interface Inner{
      int x;
   }
}
```

Вопрос 6.3.
Дан код:

```
interface Inter1{
   void f();
}
interface Inter2 extends Inter1{
   void f();
}
class Cl1 implements Inter1{
   public void f(){
      System.out.println("one");
   }
}
class Cl2 implements Inter2{
   public void f(){
      System.out.println("two");
   }
}
```

Какой метод f() будет вызван при выполнении следующего кода (1)?

```
Inter2 obj = new Cl2();
((Inter1) obj).f();
```

1) метод f() класса Cl1
2) метод f() класса Cl2
3) произойдет ошибка компиляции, т. к. ссылка типа Inter2 не может ссылаться на объект Cl2
4) произойдет ошибка компиляции, т. к. ссылку obj1 нельзя привести к типу Inter1

Вопрос 6.4.
Укажите, какое ключевое слово используется для объявления аннотации:
1) @interface;
2) @class;
3) @annotation;
4) @custom_annotation.

Ответы к главе 6

Вопрос 6.1.
Метод **returner()** класса **Quest1** реализует соответствующий абстрактный метод класса **Quest100** и одновременно метод **returner()** интерфейса **Quest10**. В последнем случае возвращаемый тип заменяется его подклассом (метод подставки).
Ответ: 4.

Вопрос 6.2.
Поля интерфейса имеют модификаторы **public final static** и должны быть инициализированы при объявлении.
Ответ: 3,4.

Вопрос 6.3.
На объект **obj** класса **Cl2** ссылается ссылка базового типа (интерфейса) **Inter1**, метод **f()** для класса **Cl2** переопределен, следовательно, при выполнении кода механизм позднего связывания вызовет метод **f()** класса **Cl2.**
Ответ: 2.

Вопрос 6.4.
Для объявления аннотации используется объявление вида **@interface**
Ответ: 1.

Ответы:
6.1. — 4 6.3. — 2
6.2. — 3,4 6.4. — 1

Тестовые задания к главе 7

Вопрос 7.1.

Дан код**:**

```
String s1 = "Minsk";
String s2 = new String("Minsk");
if(s1.equals(s2.intern())){
   System.out.print("true");
} else {
   System.out.print("false");
}
if(s1 == s2){
   System.out.print("true");
} else{
   System.out.print("false");
}
```

В результате при компиляции и запуске будет выведено (1):

1) truefalse
2) falsetrue
3) truetrue
4) falsefalse
5) ошибка компиляции: заданы некорректные параметры для метода equals()

Вопрос 7.2.

Дан код**:**

```
StringBuilder sb1 = new StringBuilder("I like Java.");//1
StringBuilder sb2 = new StringBuilder(sb1);//2
if (sb1.equals(sb2)){
   System.out.println("true");
} else {
   System.out.println("false");
}
```

В результате при компиляции и запуске будет выведено (1):

1) true
2) false
3) ошибка компиляции в строке 1
4) ошибка компиляции в строке 2

Вопрос 7.3.

Что будет результатом компиляции и выполнения следующего кода?

```
Pattern p = Pattern.compile("(1*)0");
Matcher m = p.matcher("111110");
System.out.println(m.group(1));
```

1) вывод на консоль строки «**111110**»;
2) вывод на консоль строки «**11111**»;
3) ошибка компиляции;
4) ошибка выполнения.

Вопрос 7.4.

Что выведется на консоль при компиляции и выполнении следующих строчек кода (1)?

```
Locale loc = new Locale("ru", "RU");
System.out.println(loc.getDisplayCountry(Locale.US));
```

1) United States;
2) Соединенные Штаты;
3) Россия;
4) Russia;
5) ошибка компиляции.

Вопрос 7.5.
Укажите, какому пакету принадлежат классы, позволяющие форматировать числа и даты: NumberFormat и DateFormat.
1) java.text
2) java.util.text
3) java.util
4) java.lang

Ответы к главе 7

Вопрос 7.1.
Синтаксис языка Java позволяет создавать строковые объекты с использованием упрощенного синтаксиса, но в этом случая строка автоматически размещается в пуле литералов. Метод **intern()** класса **String** размещает строку в пуле литералов и возвращает ссылку на нее, ссылку нужно сохранить для дальнейшего использования.
Ответ: 1.

Вопрос 7.2.
Для класса **StringBuffer** не переопределены методы **equals()** и **hashCode()**.
Ответ: 2.

Вопрос 7.3.
Начальное состояние объекта типа **Matcher** неопределенно, поэтому до вызова метода **group()** на объекте необходимо вызвать любой из методов обработчиков (например, **find()** или **lookingAt()).**
Ответ: 4.

Вопрос 7.4.
В результате компиляции и запуска должно выводиться название страны для локали ("ru","RU") так, как оно пишется в локали US.
Ответ: 4.

Вопрос 7.5.
Классы **NumberFormat** и **DateFormat** расположены в пакете **java.text.**
Ответ: 1.

Ответы:
7.1 — 1 7.3 — 4 7.5 — 1
7.2 — 2 7.4 — 4

Тестовые задания к главе 8

Вопрос 8.1.
Выберите правильные утверждения (3):
1) Проверяемые (checked) исключения являются наследниками класса java.lang.Exception
2) Непроверяемые (unchecked) исключения являются наследниками класса java.lang.Error
3) Непроверяемые (unchecked) исключения являются наследниками класса java.lang.Exception
4) Проверяемые (checked) исключения обязательно обрабатываются
5) Непроверяемые (unchecked) исключения невозможно обработать

Вопрос 8.2.
Дан код**:**

```
try {
   FileReader fr1 = new FileReader("test1.txt");
   try {
      FileReader fr2 = new FileReader("test2.txt");
   } catch (IOException e) {
      System.out.print("test2");
   }
   System.out.print("+");
} catch (FileNotFoundException e) {
   System.out.print("test1");
}
System.out.print("+");
```

Какая строка выведется на консоль при компиляции и запуске этого кода,
если файл test1.**txt** существует и доступен, а test2.**txt** нет (1)?
1) test1
2) test1+
3) test1++
4) test2
5) test2+
6) test2++
7) ошибка компиляции

Вопрос 8.3.
Дана иерархия исключений:

```
class A extends java.lang.Exception{}
class B extends A{}
class C extends B{}
class D extends A{}
class E extends A{}
class F extends D{}
class G extends D{}
class H extends E{}
```

Выберите цепочки блоков catch, использование которых не приведет
к ошибке компиляции, если в соответствующем блоке try могут генерироваться исключения типа C**,**D**,**G**,**H (3):
1) **catch**(C e){}**catch**(D e){} **catch**(H e){}**catch**(**A** e){}
2) **catch**(C e){}**catch**(D e){}**catch**(E e){}**catch**(A e){}
3) **catch**(C e){}**catch**(D e){}**catch**(G e){}**catch**(A e){}
4) **catch**(A e){}**catch**(D e){}**catch**(G e){}**catch**(H e){}
5) **catch**(E e){}**catch**(D e){}**catch**(B e){}**catch**(A e){}

Вопрос 8.4.
Дан код:

```
class A{
   public void f() throws IOException{}
}
class B extends A{}
```

Каким образом можно переопределить метод f() в классе **B**, не вызвав при
этом ошибку компиляции (4)?
1) **public void** f() **throws** Exception {}
2) **public void** f() **throws** IOException {}
3) **public void** f() **throws** InterruptedException, IOException {}
4) **public void** f() **throws** IOException, FileNotFoundException {}
5) **public void** f() **throws** FileNotFoundException {}
6) **public void** f() **throws** FileNotFoundException, InternalError {}

Вопрос 8.5.
Дан код**:**

```
public class Quest {
   private int qQ;
   public Quest(int q) {
      qQ = 12 / q;//1
   }
   public int getQQ() {
      return qQ;//2
   }
   public static void main(String[] args) {
      Quest quest = null;
      try {
         quest = new Quest(0);//3
      } catch (Exception e) {//4 }
   System.out.println(quest.getQQ());//5
   }
}
```

Укажите строку, выполнение которой приведет к необрабатываемой в данном коде исключительной ситуации (1):
1) **1**
2) **2**
3) **3**
4) 4
5) 5

Ответы к главе 8

Вопрос 8.1.
Проверяемые исключения в Java являются наследниками класса **Exception**, непроверяемые — наследниками класса **RuntimeException**, который, в свою очередь, наследуется от **java.lang.Exception**. Проверяемые исключения обязательно в коде обрабатывать либо в секции **try-catch**, либо с помощью ключевого слова **throws**. Непроверяемые
исключения можно обрабатывать в коде так же, как и проверяемые.
Ответ: 1,3,4.

Вопрос 8.2.
При выполнении оператора **FileReader fr2 = new FileReader("test2.txt");** возникнет исключение **FileNotFoundException** во вложенном блоке **try**, которое является
производным от **IOException**.
Ответ: 6.

Вопрос 8.3.
Подклассы исключений в блоках **catch** должны следовать перед любым из их суперклассов, иначе суперкласс будет перехватывать эти исключения.
Ответ: 1,2,5.

Вопрос 8.4.
Переопределяемый метод в подклассе не может содержать в инструкции **throws**
исключений, необрабатываемых в соответствующем методе суперкласса. Это относится только к checked-исключениям.
Ответ: 2,4,5,6.

Вопрос 8.5.
Если в конструкторе выбрасывается исключение, то объект не создается.
ArithmeticException гасится соответствующим блоком **catch**, не обработается только
NullPointerException, полученный в строке 5.
Ответ: 5.

Ответы:
8.1. — 1,3,4 8.3. — 1,2,5 8.5. — 5
8.2. — 6 8.4. — 2,4,5,6

Тестовые задания к главе 9

Вопрос 9.1.
Укажите классы, которые стоят во главе символьной иерархии потоков ввода/вывода в **Java** (2):
1) InputStreamReader
2) OutputStreamWriter
3) Reader
4) Writer
5) InputStream
6) OutputStream

Вопрос 9.2.
Дан код:

```
public class Quest {
   public static void main(String[] args) throws IOException {
      Scanner sc1 = new Scanner(System.in);
      int x1 = 0;
      x1 = sc1.nextInt();
      sc1.close();
      int x2=0;
      x2 = System.in.read();
      System.out.println(x1+" "+(char)x2);
   }
}
```

Что будет результатом компиляции и запуска этого кода, если предполагается при первом считывании с консоли ввести 1, а при втором — 2 (1)?
1) компиляция и запуск пройдут успешно, на консоль выведется строка «1 2»;
2) компиляция и запуск пройдут успешно, на консоль выведется строка «1 0»;
3) компиляция и запуск пройдут успешно, на консоль выведется строка «1»;
4) компиляция кода осуществится успешно, а при работе программы возникнет исключительная ситуация;
5) ошибка компиляции.

Вопрос 9.3.
Дан код:
```
class A {
   public int a=0;
   public A() {
      a = 1;
   }
}
class B extends A implements Serializable {
   public int b=0;
   public B() {
      a = 2;
      b = 3;
   }
}
```
Какие значения примут поля объекта, созданного от класса B с помощью
конструктора без параметров, на который ссылается ссылка b при десериализации (1)?
1) b.**a**=1, b.b=3
2) b.**a**=2, b.b=3
3) b.**a**=0, b.b=0
4) b.**a**=0, b.b=3
5) b.**a**=1, b.b=0
6) десериализация невозможна, так как класс **A** не реализует интерфейс Serializable

Вопрос 9.4.
Укажите классы байтовых потоков ввода-вывода, для которых не существует аналогичных классов в символьной иерархии потоков ввода-вывода:
1) DataInputStream
2) InputStreamReader
3) ObjectOutputStream
4) StringWriter

Вопрос 9.5.
Компиляция каких строк следующего кода приведет к ошибке (3)?
```
public class Quest {
   public static void main(String[] args) throws IOException {
      File file = new File("files.t1.txt");
      FileWriter obj1 = new FileWriter(file); //1
      ByteArrayInputStream obj2 = new ByteArrayInputStream(file); //2
      InputStreamReader obj3 = new InputStreamReader(file); //3
      BufferedReader obj4 = new BufferedReader(file); //4
      PrintWriter obj5 = new PrintWriter(file); //5
   }
}
```
1) ошибка компиляции в строке 1;
2) ошибка компиляции в строке 2;
3) ошибка компиляции в строке 3;
4) ошибка компиляции в строке 4;
5) ошибка компиляции в строке 5.

Ответы к главе 9
Вопрос 9.1.
На вершине иерархии байтовых потоков ввода-вывода два **абстрактных** класса:
InputStream и **OutputStream**. На вершине иерархии символьных потоков ввода-вывода два **абстрактных** класса: **Reader** и **Writer**. **InputStreamReader** — поток ввода, который переводит байты в символы. **OutputStreamWriter** — поток ввода, который переводит символы в байты.
Ответ: 3,4.

Вопрос 9.2.
При закрытии потока, на который указывает ссылка **sc1**, закроется и поток **System.in**. Дальнейшее считывание с помощью него становится невозможным.
Ответ: 4.

Вопрос 9.3.
При *десериализации* производного *класса, наследуемого от несериализуемого класса*, вызывается *конструктор без параметров* родительского несериализуемого класса.
Ответ: 1.

Вопрос 9.4.
Классы **InputStreamReader** и **StringWriter** относятся к символьным потокам ввода-вывода.
Ответ: 1, 3.

Вопрос 9.5.
Для классов **ByteArrayInputStream**, **InputStreamReader**, **BufferedReader** не предусмотрены конструкторы, принимающие параметр типа **File**.
Ответ: 2, 3, 4.

Ответы:
9.1. — 3,4 9.3. — 1 9.5. — 2,3,4
9.2. — 4 9.4. — 1,3

Тестовые задания к главе 10

Вопрос 10.1.
Какой класс коллекции позволяет наращивать и сокращать размер, предоставляет индексный доступ к элементам, но его методы несинхронизированы (1)?
1) java.util.HashSet
2) java.util.ArrayList
3) java.util.LinkedHashSet

Вопрос 10.2.
Дан класс:
```
class X{
   private int x;
   public X(int x){this.x = x;}
   public int hashCode(){
      return 2;
   }
}
```
Каков будет размер коллекции set после выполнения следующего кода (1)?
```
X obj1 = new X(1); X obj2 = new X(1);
Set<X> set = new HashSet<X>();
set.add(obj1);
set.add(obj2);
```
1) **0**;
2) **2**;
3) **1**;
4) при выполнении данного кода возникнет исключительная ситуация.

Вопрос 10.3.
Даны два фрагмента кода:
1.
```
Vector<String> vct = new Vector<String>();
vct.add("One"); vct.add("Two");
vct.add("Three"); vct.add("Four");
vct.add("Five"); vct.add("Six");
Iterator itr = vct.iterator();
vct.remove(0);
System.out.println(itr.next());
```
2.
```
Vector<String> vct = new Vector<String>();
vct.add("One"); vct.add("Two");
vct.add("Three"); vct.add("Four");
vct.add("Five"); vct.add("Six");
Enumeration enm = vct.elements();
vct.remove(0);
System.out.println(enm.nextElement());
```
Укажите разницу при выполнении первого и второго фрагмента кода (1):
1) при выполнении первого фрагмента на консоль выведется строка «Two»,
а при выполнении второго произойдет исключительная ситуация;
2) при выполнении второго фрагмента на консоль выведется строка «Two»,
а при выполнении первого произойдет исключительная ситуация;
3) выполнение двух фрагментов кода приведет к исключительной ситуации;
4) при выполнении двух фрагментов кода на консоль выведется строка «Two»;
5) при компиляции первого фрагмента приведет к ошибке;
6) компиляция второго фрагмента кода приведет к ошибке;
7) компиляция двух фрагментов кода приведет к ошибке.

Вопрос 10.4.
Дан код:
```
import java.util.*;
enum PCounter {UNO, DOS, TRES, CUATRO, CINCO, SEIS, SIETE};
public class Quest {
   public static void main(String[] args) {
      EnumSet<PCounter> enst1 = EnumSet.range(PCounter.TRES, PCounter.CINCO);//1
      EnumSet<PCounter> enst2 = EnumSet.complementOf(enst1);//2
      System.out.println(enst2);
   }
}
```
Что будет выведено при попытке компиляции и запуска программы (1)?
1) [UNO, DOS, TRES, CINCO, SEIS, SIETE]
2) [DOS, TRES, CUATRO, CINCO, SEIS]
3) [UNO, DOS, **SEIS**, **SIETE**]
4) ошибка компиляции в строке 1
5) ошибка компиляции в строке 2
6) ничего из перечисленного

Вопрос 10.5.
Что будет результатом компиляции и запуска следующей программы (1)?

```
import java.util.*;
public class Quest {
   public static void main(String[] args) {
      NavigableMap<String, Number> nmap = new TreeMap<String, Number>();
      nmap.put("one", new Integer(1)); nmap.put("two", new Integer(2));
      nmap.put("three", new Integer(3)); nmap.put("four", new Integer(4));
      Map<String, Number> map = nmap.headMap("three");
      System.out.println(map);
   }
}
```

1) {four=4, one=1}
2) {one=2, two=2}
3) {three=3, four=4}
4) {four=4, one=1, three=3}
5) {one=2, two=2, three=3}

Ответы к главе 10

Вопрос 10.1.
Классы **HashSet** и **LinkedHashSet** не предоставляют индексный доступ к хранимым элементам.
Ответ: 2.

Вопрос 10.2.
Интерфейс **Set** уникальность хранимых объектов определяет реализацией метода **equals()**.
Ответ: 2.

Вопрос 10.3.
Использование **Iterator** более безопасно, чем **Enumeration**, поскольку **Iterator** не позволяет менять объект коллекции иным способом, кроме как с использованием собственного метода **remove()**, а при попытке изменения объекта коллекции выбрасывает исключение **ConcurrentModificaitonException**.
Ответ: 2.

Вопрос 10.4.
Вызовом метода **EnumSet<T> complementOf(EnumSet<T> s) —** создается множество, содержащее все элементы, которые отсутствуют в указанном множестве **s.**
Ответ: 3.

Вопрос 10.5.
По умолчанию **TreeMap** сортирует элементы по ключу по возрастанию, метод **headMap()** с одним параметром возвращает элементы с начала набора до указанного элемента, не включая его.
Ответ: 1.

Ответы:
10.1. — 2 10.3. — 2 10.5. — 1
10.2. — 2 10.4. — 3

Тестовые задания к главе 11

Вопрос 11.1.

Дан класс:

```
class InThread implements Runnable{
   public void run() {
      System.out.println("running...");
   }
}
```

Укажите правильные варианты создания потокового объекта (1):

1) **new** Thread().**new** InThread();
2) **new** Runnable(**new** InThread());
3) **new** Thread(Intread);
4) **new** Thread(**new** InThread());
5) **new** InThread().

Вопрос 11.2.

Укажите методы, определенные в классе java.lang.Thread (4):

1) join()
2) getPrioroty()
3) wait()
4) notifyAll()
5) sleep()
6) getName()

Вопрос 11.3.

Укажите состояния потока, при вызове на которых метод isAlive() класса java.lang.Thread вернет значение true (4):

1) *NEW*
2) RUNNABLE
3) BLOCKED
4) WAITING
5) TIMED_WAITING
6) TERMINATED

Вопрос 11.4.

Дан код:

```
class InThread implements Runnable{
   public void run() {System.out.println("running..."); }
}
public class Quest {
   public static void main(String[] args) {
      ExecutorService exec = Executors.newFixedThreadPool(2);
      exec.execute(new InThread()); exec.execute(new InThread());
      exec.execute(new InThread()); exec.execute(new InThread());
      exec.execute(new InThread()); exec.execute(new InThread());
      exec.shutdown(); while (!exec.isTerminated()) { }
   }
}
```

Сколько потоков выполнит объект exec при запуске этого кода (1)?

1) *2*
2) 4
3) *0*
4) 6
5) столько, сколько успеет до завершения метода main()

Вопрос 11.5.
Дан класс Lamp(лампочка). Расставьте указанные ниже строки кода метода turnOn() так, чтобы не допустить ситуации включения лампочки, когда она уже включена (поле lamp имеет значение true, когда лампочка включена):

```
class Lamp {
   private boolean lamp = false;
   public synchronized void turnOn() throws InterruptedException {
_________________
   }
}
```

a) lamp = **true**;
b) notify();
c) **while** (lamp == **true**) wait();
Выберите один вариант (1):
1) abc;
2) bca;
3) acb;
4) cba;
5) cab.

Ответы к главе 11

Вопрос 11.1.
Потоковым объектом является объект класса **Thread**. Чтобы создать потоковый объект с помощью объекта, класс которого реализует интерфейс **Runnable**, необходимо придать этому объекту функциональность класса **Thread** — передать объект **Runnable** в конструктор класса **Thread**.
Ответ: 4.

Вопрос 11.2.
Методы **wait()** и **notifyAll()** определены в классе **java.lang.Object**. Остальные методы определены в классе **java.lang.Thread**.
Ответ: 1, 2, 5, 6.

Вопрос 11.3.
Метод **isAlive()** возвращает **true**, когда поток находится в работоспособном состоянии или в состоянии, из которого он может вернуться в работоспособное.
Ответ: 2, 3, 4, 5.

Вопрос 11.4.
С помощью объекта, представляющего собой пул потоков фиксированного размера, выполняются все шесть запускаемых из метода **main()** потоков, по два одновременно.
Ответ: 4.

Вопрос 11.5.
Последовательность действий для метода **turnOn()** следующая: если лампочка горит — ждем, пока ее не выключат, включаем лампочку, оповещаем возможных остальных ждущих.
Ответ: 5.

Вопрос 11.6.
Класс **ReentrantLock** находится в пакете **java.util.concurrent.locks**, класс **AtomicInteger** в пакете **java.util.concurrent.atomic**. Остальные классы и интерфейсы находятся в пакете **java.util.concurrent**.
Ответ: 1, 3, 4, 5.

Ответы:

11.1. — 4	11.3. — 2,3,4,5	11.5. — 5
11.2. — 1,2,5,6	11.4. — 4	11.6. — 1,3,4,5

Тестовые задания к главе 12

Вопрос 12.1.

Какие пакеты содержат интерфейсы и классы JDBC (1)?

1) java.db и javax.db
2) java.sql и javax.sql
3) java.jdbc.sql и javax.jdbc.sql
4) org.sql и org.jdbc
5) java.jdbc и javax.jdbc

Вопрос 12.2.

Даны операторы языка Java:

a) Connection cn =DriverManager.**getConnection**("jdbc:mysql://localhost:3306", "root", "pass");
b) ResultSet rs = st.executeQuery("SELECT * FROM users WHERE id=1");
c) Class.**forName**("org.gjt.mm.mysql.Driver");
d) Statement st = cn.createStatement();
e) System.**out**.println(rs.next());

Расставьте их в правильной последовательности так, чтобы с помощью получившегося кода можно было извлечь данные из БД (1):

1) abdec;
2) cadbe;
3) cabde;
4) ebdac;
5) abced.

Вопрос 12.3.

Какие типы statement-объектов существуют в JDBC (3)?

1) Statement
2) ResultStatement
3) PreparedStatement
4) DriverStatement
5) MetaDataStatement
6) CallableStatement

Вопрос 12.4.
Дана база данных test с таблицей users:

id	name
1	*Ivanov*

Какая информация добавится в таблицу users после выполнения следующего кода (2)?
Connection cn = /* корректное получение соединения */;
Statement st = cn.createStatement();
st.executeUpdate("INSERT INTO users VALUES (2, 'Petrov')");
cn.setAutoCommit(false);
JDBC
st.executeUpdate("INSERT INTO users VALUES (3, 'Sidorov')");
cn.setSavepoint("point1");
st.executeUpdate("INSERT INTO users VALUES (4, 'Vasechkin')");
cn.rollback();
st.executeUpdate("INSERT INTO users VALUES (5, 'Blinov')");
cn.commit();
1) id=2, name=Petrov
2) id=3, name=Sidorov
3) id=4, name=Vasechkin
4) id=5, name=Blinov

Вопрос 12.5.
Укажите, каким способом можно выполнить хранимую процедуру с помощью JDBC (3):
1) вызвать метод execute() на объекте CallableStatement
2) вызвать метод executeQuery() на объекте CallableStatement
3) вызвать метод executeUpdate() на объекте CallableStatement
4) вызвать метод executeProcedure() на объекте CallableStatement
5) вызвать метод execute() на объекте StoredStatement
6) вызвать метод executeQuery() на объекте StoredStatement
7) вызвать метод executeUpdate() на объекте StoredStatement
8) вызвать метод executeProcedure() на объекте StoredStatement

Ответы к главе 12

Вопрос 12.1.
Классы и интерфейсы JDBC находятся в пакетах **java.sql** и **javax.sql.**
Ответ: 2.

Вопрос 12.2.
Последовательность действий для подключения к БД с помощью JDBC следующая:
загрузка драйвера, установление соединения, создание объекта для выполнения запросов, выполнение запроса, обработка результатов, закрытие открытых соединений.
Ответ: 2.

Вопрос 12.3.
В API JDBC существует три интерфейса, реализации которых могут предоставлять statement-объекты: **Statement**, **PreparedStatement**, **CallableStatement.**
Ответ: 1, 3, 6.

Вопрос 12.4.
Метод **rollback()**, выполненный на объекте типа **Connection**, откатит состояние БД до выполнения последней операции commit; в таблицу добавится информация (2, Petrov) и (5, Blinov).
Ответ: 1, 4.

Вопрос 12.5.
Хранимую процедуру можно выполнить на объекте **CallableStatement**, после установки входных и выходных параметров используются методы **execute(), executeQuery()**
или **executeUpdate().**
Ответ: 1, 2, 3.

Ответы:
12.1. — 2 12.3. — 1,3,6 12.5. — 1,2,3
12.2. — 2 12.4. — 1,4

Тестовые задания к главе 13

Вопрос 13.1.
Выберите правильные утверждения (2):
1) протокол TCP не гарантирует доставку пакетов;
2) протокол UDP не гарантирует доставку пакетов;
3) протоколы TCP и UDP используются в сервисах, где важна гарантированная доставка сообщений;
4) протоколы TCP и UDP используются в сервисах, где важна быстрая доставка сообщений;
5) протокол TCP используется в сервисах, где важна гарантированная доставка сообщений, протокол UDP используется в сервисах, где важна быстрая доставка сообщений;
6) протокол UDP используется в сервисах, где важна гарантированная доставка сообщений, протокол TCP используется в сервисах, где важна быстрая доставка сообщений.

Вопрос 13.2.
IP сервиса google.ru 173.194.35.184. Укажите, с помощью каких методов
класса java.net.InetAddress можно это выяснить (2):
1) getHostAddress()
2) getCanonicalHostName()
3) getAddress()
4) getHostName()
5) getLocalHost()

Вопрос 13.3.
Дан код:
```
try {
   ServerSocket server = new ServerSocket(1234);
   Socket sock = server.accept();
} catch (UnknownHostException e) {
   e.printStackTrace();
} catch (IOException e) {
   e.printStackTrace();
}
```
Укажите, скольких клиентов может обслужить запущенный таким образом сервер (1):
1) ни одного, произойдет ошибка компиляции;
2) сколько угодно;
3) сколько угодно, но не больше, чем разрешено подключений на порт 1234;
4) одного клиента.

Вопрос 13.4.
Укажите, какая исключительная ситуация возникает при выполнении оператора
Socket **s** = *new* Socket("15.12.14.16", 3456); в случае, если соединение
с сервером не установлено (1):
1) MalformedURLException
2) UnknownHostException
3) UnknownURLException
4) ConnectException
5) UnknownPortException

Вопрос 13.5.
Дан код:

```
public class Quest {
   public static void main(String[] args) {
      new Thread() {
         public void run() {
         MyServer.main();
      }
   }.start();
      new Thread() {
         public void run() {
            try {
               sleep(1000);
               MyClient.main();
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }
      }.start();
   }
}
class MyServer {
   public static void main() {
      try {
         ServerSocket server = ServerSocketFactory.getDefault().createServerSocket(12345);
         Socket socket = server.accept();
         Thread.sleep(1000);
         server.close();
         socket.close();
      } catch (IOException e) {
         System.out.println("IOException on server.");
      } catch (InterruptedException e) {
         System.out.println("InterruptedException on server.");
      }
   }
}
class MyClient {
   public static void main() {
      try {
         Socket s = SocketFactory.getDefault().createSocket("localhost",12345);
         System.out.print(s.isConnected()); Thread.sleep(5000);
         System.out.println(s.isConnected());
      } catch (IOException e) {
         System.out.println("IOException on client.");
      } catch (InterruptedException e){
         System.out.println("InterruptedException on client.");
      }
   }
}
```

Какая строка выведется на консоль в результате запуска этого кода, если исключительных ситуаций при его работе не было (1)?

1) truetrue
2) truefalse
3) falsetrue
4) falsefalse
5) нет правильного

Ответы к главе 13

Вопрос 13.1.
TCP — протокол образует постоянное соединение и гарантирует доставку пакета.
UDP не образует постоянного соединения и не гарантирует доставку пакета, однако делает это быстро.
Ответ: 2, 5.

Вопрос 13.2.
IP адрес ресурса можно узнать с помощью методов **getHostAddress()**, **getAddress**().
getLocalHost() — возвратит номер хоста текущей машины.
Ответ: 1, 3.

Вопрос 13.3.
Метод **accept()** не возвращает управление, пока не подключится клиент.
Ответ: 4.

Вопрос 13.4.
Если соединение с сервером не установлено, возникает исключение типа **ConnectException**.
Ответ: 4.

Вопрос 13.5.
Метод **isConnected()** говорит о том, что сокет был соединен с удаленной стороной. Информацию о том, был ли данный сокет закрыт, этот метод не предоставляет.
Ответ: 1.

Ответы:
13.1. — 2,5 13.3. — 4 13.5. — 1
13.2. — 1,3 13.4. — 4

Тестовые задания к главе 14

Вопрос 14.1.
Укажите параметры, которые можно использовать при объявлении документа xml (3):
1) errorPage
2) version
3) schema
4) encoding
5) charset
6) standalone

Вопрос 14.2.
Укажите вид документа XML, который соответствует следующему описанию: xml-документ следует синтаксическим правилам XML и правилам, определенным в его DTD или в XSD (1):
1) неправильный xml-документ;
2) правильный xml-документ;
3) неправильно-форматированный xml-документ.

Вопрос 14.3.
Укажите классы, которые входят в состав пакета javax.xml.parsers (4):
1) XMLInputFactory
2) XMLStreamReader
3) DocumentBulider
4) DocumentBuilderFactory
5) Document
6) SAXParser
7) SAXParserFactory

Вопрос 14.4.
Укажите утилиту java-платформы, с помощью которой возможно обратное создание на основе XML-схемы классов (1):
1) java;
2) rmic;
3) apt;
4) xjc.

Вопрос 14.5.
Укажите, какие события обрабатывает **SAX**-парсер (1):
1) событие начало документа;
2) событие конец документа;
3) событие текстовые данные;
4) событие начало элемента;
5) событие конец элемента;
6) все перечисленные события можно обработать.

Ответы к главе 14

Вопрос 14.1.
Параметры xml-документа: **version** — номер версии XML, **encoding** — кодировка, **standalone** — определяет автономность XML.
Ответ: 2, 4, 6.

Вопрос 14.2.
Неправильные документы не следуют синтаксическим правилам XML. Правильные документы следуют синтаксическим правилам XML и правилам, определенным в их DTD или в XSD. Документы форматированные правильно не имеют DTD или XSD.
Ответ: 2.

Вопрос 14.3.
JAXP располагается в пакете **javax.xml.parsers**, в состав которого входят четыре класса: **DocumentBuilder**, **DocumentBuilderFactory**, **SAXParser**, **SAXParserFactory**.
Ответ: 3, 4, 6, 7.

Вопрос 14.4.
Возможно обратное создание на основе XML-схемы классов на языке Java с помощью команды **xjc.exe**.
Ответ: 4.

Вопрос 14.5.
К событиям, которые обрабатывает синтаксический анализатор SAX, относятся события начала документа, начала элемента, пары значений атрибута, текстовые данные, конец элемента и конец документа.
Ответ: 6.

Ответы:
14.1. — 2,4,6 14.3. — 3,4,6,7 14.5. — 6
14.2. — 2 14.4. — 4

Тестовые задания к главе 15

Вопрос 15.1.

Параметры конфигурации сервлета были заданы в файле-дескрипторе развертывания сервлета. Указать, с помощью методов какого интерфейса можно
прочитать эти параметры в сервлете (1):

1) Servlet
2) ServletRequest
3) ServletResponse
4) ServletConfig
5) ServletContext

Вопрос 15.2.

Выбрать вариант URL, определяемый в адресной строке браузера и отправляющий http-запрос сервлету методом **POST** (1)**:**

1) http://localhost:8080/Quests/Quest?method=post
2) POST http://localhost:8080/Quests/Quest
3) [POST]http://localhost:8080/Quests/Quest
4) http://localhost:8080/Quests/Quest[post]
5) ничего из вышеперечисленного

Вопрос 15.3.

Выбрать метод, который за жизненный цикл сервлета может быть вызван
более одного раза (1):

1) init()
2) service()
3) destroy()
4) конструктор сервлета

Вопрос 15.4.

Дан код:

```
public class Quest extends HttpServlet {
   protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException, IOException {
      String str1 = request.getParameter("param1");
      String str2 = request.getParameter("param2");
      response.getWriter().println(str1 + " " + str2);
   }
}
```

Указать вариант записи URL, в результате запроса которого в сервлет инициализирует переменную str1 в значение one, а переменную str2 в значение
two**:**

1) http://localhost:8080/Tests3-18/Quest?param1=\"one\"&param2=\"two\"
2) http://localhost:8080/Tests3-18/Quest?param1="one"&param2="two"
3) http://localhost:8080/Tests3-18/Quest?param1=one&param2=two
4) http://localhost:8080/Tests3-18/Quest?param1=one+param2=two
5) http://localhost:8080/Tests3-18/Quest?param1=one?param2=two
6) http://localhost:8080/Tests3-18/Quest?param1="one"?param2="two"

Ответы к главе 15

Вопрос 15.1.

Центральной абстракцией API сервлета является интерфейс **Servlet**. Интерфейс **ServletRequest** инкапсулирует связь клиента с сервером. Интерфейс **ServletResponse** инкапсулирует обратную связь сервлета с клиентом. Интерфейс **ServletContext** используется для взаимодействия с контейнером сервлетов. Интерфейс **ServletConfig** представляет собой конфигурацию сервлета, используется, в основном, на этапе инициализации.

Ответ: 4.

Вопрос 15.2.
Послать http-запрос из адресной строки браузера методом **POST** нельзя**.**
Ответ: 5.

Вопрос 15.3.
Веб-контейнер создает объект сервлета, вызывает метод **init()**. Метод **service()** вызывается в сервлете для управления каждым запросом. Перед уничтожением объекта
контейнер вызовет метод **destroy().**
Ответ: 2.

Вопрос 15.4.
Запрос от параметров отделяется символом «?», параметры между собой — символом **«&».**
Ответ: 3.

Ответы:
15.1. — 4 15.3. — 2
15.2. — 5 15.4. — 3

Тестовые задания к главе 16

Вопрос 16.1.
Расставьте фазы жизненного цикла **JSP** в правильном порядке (1):
a) вызов метода jspInit; b) трансляция; c) вызов метода _jspService; d) компиляция; e) загрузка класса; f) вызов метод jspDestroy; **g**) создание экземпляра.
1) dbcgaef;
2) dbgeacf;
3) bdegacf;
4) bdcgeaf.

Вопрос 16.2.
Укажите, в чем состоит различие директивы include (<%@ include … %>)
и action-тэг include (<jsp:include />) (1):
1) различий нет, это две формы записи, использующие **JSP** и XML синтаксис соответственно;
2) директива include позволяет вставлять текст или код в процессе трансляции страницы JSP в сервлет, однако, в отличие от действия jsp**:**include, рассматривает ресурс как статический объект;
3) директива include позволяет вставлять текст или код в процессе трансляции страницы JSP в сервлет, однако, в отличие от действия jsp**:**include, рассматривает ресурс как динамический объект.

Вопрос 16.3.
Укажите объекты, доступные в коде **JSP** без их специального объявления или импорта (5):
1) **out**
2) request
3) response
4) session
5) pageContext
6) cookies

Вопрос 16.4.
Выберите идентификаторы **JSP**-директив (3):
1) page
2) useBean
3) forward
4) include
5) taglib
6) param

Ответы к главе 16

Вопрос 16.1.
Жизненный цикл JSP включает следующие шаги (фазы): трансляция страницы JSP — создание исходного кода сервлета; компиляция страницы JSP — компилятор переведет полученный код в байт-код; загрузка класса; создание экземпляра; вызов метода **jspInit()**; вызов метода **_jspService()** для каждого запроса; вызов метода **jspDestroy().**
Ответ: 3.

Вопрос 16.2.
Форма **<jsp:directive.include file="URL" />** JSP представляет вариант записи директивы **<%@ include %>**. Эта директива рассматривает ресурсы как статические объекты, в отличие от одноименного действия. При использовании действия **jsp:include** нет
возможности использовать во вставляемом файле код JSP.
Ответ: 2.

Вопрос 16.3.
К неявным JSP-объектам относят: **request**, **response**, **out**, **pageContext**, **session**, **application**, **config**, **page**, **exception**.
Ответ: 1, 2, 3, 4, 5.

Вопрос 16.4.
В jsp определены три основные директивы: **page**, **include**, **taglib**.
Ответ: 1, 4, 5.

Ответы:
16.1. — 3 16.3. — 1,2,3,4,5
16.2. — 2 16.4. — 1,4,5

Тестовые задания к главе 17

Вопрос 17.1.
Выберите корректные утверждения (2):
1) сессия может использоваться разными сервлетами для доступа к одному клиенту;
2) сессия может быть завершена только автоматически;
3) сессия не может быть повторно завершена;
4) если промежуток активного времени для сессии установить в ноль или отрицательное число, то сессию невозможно будет создать.

Вопрос 17.2.
Дан код**:**

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response) throws ServletException, IOException {
   HttpSession session = request.getSession(false);
   if (session == null) {
      response.getWriter().println("no session");
      session = request.getSession(true);
  } else {
     // i - статическое поле сервлета; static int i = 1 при объявлении
      response.getWriter().println("session " + i);
      i++;
   }
}
```

Какая информация будет результатом выполнения трехкратного запроса
к указанному методу doGet, если у клиента запрещен прием cookie-файлов (1)?

1) no session;
2) session 1;
3) session 2;
4) session 3;
5) ничего из вышеперечисленного.

Вопрос 17.3.
Дан код:

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response) throws ServletException, IOException {
   // i - статическое поле сервлета; static int i = 1 при объявлении
   Cookie[] cook = request.getCookies();
   if (cook != null) {
      Cookie cook2 = new Cookie(i + "", i + "");
      i++;
      cook2.setMaxAge(10);
      response.addCookie(cook2);
      response.getWriter().println("cookies = " + cook.length);
   } else {
      Cookie cook2 = new Cookie(0 + "", 0 + "");
      cook2.setMaxAge(10);
      response.addCookie(cook2);
      response.getWriter().println("one cookie");
   }
}
```

Укажите, сколько cookie-записей уйдет в ответе клиенту (response), если
один клиент осуществит вызов метода doGet 5 раз в течение 10 секунд (1):
1) 5;
2) 4;
3) 3;
4) 2;
5) 1;
6) 0;
7) произойдет ошибка компиляции.

Вопрос 17.4.
Укажите типы событий, которые обрабатывают listener-объекты уровня сессии, определяющие жизненный цикл сессии (3):
1) **s**ession creation
2) **a**ddition of session attributes
СЕССИИ, СОБЫТИЯ И ФИЛЬТРЫ
3) removal of session attributes
4) replacement of session attributes
5) **s**ession invalidation
6) **s**ession timeout

Вопрос 17.5.
Выберите корректные утверждения (2):
1) фильтры позволяют создавать объекты, которые могут изменять заголовок запроса;
2) фильтры не позволяют создавать объекты, которые могут изменять заголовок ответа;
3) фильтры могут создать новый запрос и ответ;
4) фильтр можно использовать для фильтрации более одного сервлета

Ответы к главе 17
Вопрос 17.1.
Сессия может использоваться разными сервлетами для доступа к одному клиенту.
Сессия пользователя может быть завершена вручную или, в зависимости от того, где запущен сервлет, автоматически. Метод **invalidate()** выдает **InvalidStateException**, если вызывается для сессии, которая была завершена. Отрицательное или нулевое значение времени жизни для сессии приводит к тому, что сессия никогда не завершается.
Ответ: 1, 3.

Вопрос 17.2.
Сессия клиента идентифицируется с помощью передачи идентификатора сессии (session id) между клиентом, осуществляющим запрос, и сервером.
Ответ: 1.

Вопрос 17.3.
Файлы Cookie хранятся на стороне клиента и отправляются клиентом серверу автоматически при осуществлении запроса.
Ответ: 5.

Вопрос 17.4.
Слушатели уровня сессии, определяющие жизненный цикл сессии — это: session creation, session invalidation, session timeout. Остальные относятся к событиям, изменяющим атрибуты сессии.
Ответ: 1,5,6.

Вопрос 17.5.
Фильтры могут изменять заголовок и содержимое запроса и ответа. Фильтры не создают запрос или ответ. Фильтр можно использовать для фильтрации более одного сервлета.
Ответ: 1, 4.

Ответы:
17.1. — 1,3 17.3. — 5 17.5. — 1,4
17.2. — 1 17.4. — 1,5,6

Тестовые задания к главе 18

Вопрос 18.1.
Укажите, какая директива используется для подключения библиотеки тегов
jstl (1):
1) jstl
2) page
3) taglib
4) include

Вопрос 18.2.
На jsp-странице используется тег <c**:out** value="{4*5+12}" />Укажите, каким будет результат обработки этого тега (1):
1) **32**;
2) 20+12;
3) 4*5+12;
4) {4*5+12};
5) ничего из вышеперечисленного.

Вопрос 18.3.
Укажите библиотеку jstl, применение тегов которой позволяет использовать
на jsp-страницы файлы свойств (*.properties) (1):
1) Core tags;
2) Formatting tags;
3) SQL tags;
4) XML tags;
5) JSTL Functions.

Вопрос 18.4.
Укажите слово-оператор, применение которого в **EL**-выражении приведет
к необходимости определения значения типа «строго более чем» (1):
1) eq
2) ne
3) lt
4) **gt**
5) le
6) ge

Вопрос 18.5.
Дан java-оператор: obj.getArrayValue(0)+obj.getArrayValue(1);, где getArrayValue —
метод, возвращающий i-й элемент инкапсулированного массива. Укажите правильное **EL**-выражение, соответствующее этому оператору (1):
1) ${obj.getArrayValue[0]+obj. getArrayValue[1]}
2) ${obj.arrayValue[0]+obj.arrayValue[1]}
3) ${obj.arrayValue(0)+obj.arrayValue(1)}
4) ${obj. getArrayValue(0)+obj. getArrayValue(1)}

Ответы к главе 18

Вопрос 18.1.
Директива **page** определяет атрибуты, относящиеся ко всей jsp-странице. Директива **include** позволяет вставлять текст или код в процессе трансляции страницы JSP в сервлет. Директива **taglib** объявляет, что данная страница JSP использует библиотеку тегов.
Ответ: 3.

Вопрос 18.2.
Выражения EL записываются внутри маркера **${ }.**
Ответ: 4.

Вопрос 18.3.
Библиотека Formatting tags(fmt) содержит теги **bundle** и **setBundle**, позволяющие обрабатывать ресурсные источники.
Ответ: 2.

Вопрос 18.4.
Значение слов-операторов: **eq** — проверка на равенство, **ne** — проверка на неравенство, **lt** — строго менее чем, **gt** — строго более чем, **le** — меньше либо равно чему-то, **ge** — больше или равно чему-то.
Ответ: 4.

Вопрос 18.5.
Оператор [ ] используется для доступа к элементам массива, списков типа **List** и отображений типа **Map**.
Ответ: 2.

Ответы:
18.1. — 3 18.3. — 2 18.5. — 2
18.2. — 4 18.4. — 4

Тестовые задания к главе 19

Вопрос 19.1.
Что должен возвращать метод doStartTag(), если в теге отсутствует тело (1)?
1) константу SKIP_BODY
2) исключение SkipBodyException
3) константу EMPTY_BODY
4) исключение EmptyBodyException
5) ничего из вышеперечисленного

Вопрос 19.2.
Какой из методов класса BodyTagSupport может быть вызван несколько раз (1)?
1) doInitBody()
2) doStartTag()
3) doAfterBody()
4) doEndTag()

Вопрос 19.3.
В файле web.xml зарегистрирована пользовательская библиотека тегов:
<taglib>
<taglib-uri>/**WEB**-INF/tld/mytaglib.tld</taglib-uri>
<taglib-location>/**WEB**-INF/tld/mytaglib.tld</taglib-location>
</taglib>
Укажите правильный вариант написания директивы, позволяющей использовать пользовательскую библиотеку тегов на jsp-странице (1):
1) <%@ taglib uri="*/tld/mytaglib.tld*" prefix="*mytag*"%>
2) <%@ taglib uri="*/WEB-INF/tld/mytaglib.tld*" prefix="*mytag*"%>
3) <%@ taglib uri="*/WEB-INF/tld/mytaglib*" prefix="*mytag*"%>
4) <%@ taglib uri="*/tld/mytaglib*" prefix="*mytag*"%>

Вопрос 19.4.
Какие значения может принимать тег <body-content> при описании пользовательского тега в tld-файле?
1) empty
2) html
3) jsp
4) no_jsp
5) tagdependent
6) choice

Вопрос 19.5.
Укажите правильный вариант подключения tld-файла при создании jspдокумента:
1) <jsp:root xmlns:jsp="*http://*java.*sun.com/JSP/Page*" xmlns:mytag="*/WEB-INF/tld/mytaglib.tld*" version="*1.2*">
2) <jsp:taglib xmlns:jsp="*http://java.sun.com/JSP/Page*" xmlns:mytag="*/WEBINF/tld/mytaglib.tld*" version="*1.2*">
3) <jsp:root xmlns:jsp="*http://java.sun.com/JSP/Page*" xmlns:uri="*/WEB-INF/tld/mytaglib.tld*" xmlms:prefix="*mytag*" version="*1.2*">
4) <jsp:taglib xmlns:jsp="*http://java.sun.com/JSP/Page*" xmlns:uri="*/WEB-INF/tld/mytaglib.tld*" xmlms:prefix="*mytag*" version="*1.2*">
5) нет правильного ответа

Ответы к главе 19

Вопрос 19.1.
Если в определении тега отсутствует тело, метод **doStartTag()** должен возвратить константу **SKIP_BODY**, дающую указание системе игнорировать любое содержимое между начальными и конечными элементами создаваемого тега.
Ответ: 1.

Вопрос 19.2.
Метод **doInitBody()** вызывается один раз перед первой обработкой тела тега, **doEndTag()** — вызывается один раз, когда отработаны все остальные методы, **doAfterBody()** — вызывается после каждой обработки тела тега. **doStartTag()** вызывается, если обнаруживается начальный элемент тега.
Ответ: 3.

Вопрос 19.3.
Директива **taglib** содержит два атрибута: **uri** (копирует значение **<taglib-uri>** в **web.xml** или прямо указывает на **tld**-файл) и **prefix**.
Ответ: 2.

Вопрос 19.4.
Поддерживаются следующие значения для **body-content**: **empty** — пустое тело; **jsp** — тело состоит из всего того, что может находиться в JSP-файле; **tagdependent** — тело интерпретируется классом, реализующим данный тег.
Ответ: 1, 3, 5.

Вопрос 19.5.
При создании jsp-документа, регистрация пользовательской библиотеки тегов происходит в корневом элементе **jsp:root**.
Ответ: 1.

Ответы:
19.1. — 1 19.3. — 2 19.5. — 1
19.2. — 3 19.4. — 1,3,5