Klausur Softwareentwicklung 1 Institut für Praktische Informatik

Klausur am 03.02.2003 Univ.-Prof. Dr. Alois Ferscha

Zuname		HS	
Vorname		Sitzplatz	
Matr. Nr.		Punkte	
SKZ		Note	

Fügen Sie fehlende Teile in den folgenden Aufgaben in die umrandeten Kästchen (Platzhalter) ein.

Hinweis: Ausschließlich die in den Kästchen angegebenen Lösungen werden berücksichtigt bzw. korrigiert. Der Rest wird ignoriert**.**

1. Operatoren und Ausdrücke (12 = 2 + 2 + 2 + 2 + 2 + 2 Punkte)

Welche Ausgaben liefert folgendes Programm? Tragen Sie die Ausgaben der einzelnen `IO.writeLn()`-Aufrufe in die daneben liegenden Felder ein.

Hinweis: Das Programm kompiliert und läuft fehlerfrei.

```
public class Operators
{
  public static void main(String args[])
  {
    int a = 5;

    IO.writeLn(a += 5);                    /* 10 */

    IO.writeLn(a--);                       /* 10 */

    IO.writeLn(++a);                       /* 10 */

    IO.writeLn((a = a + 32) % 32);         /* 10 */

    IO.writeLn(a & 15);                    /* 10 */

    IO.writeLn(a / 4);                     /* 10 */
  }
}
```

2. Typen und Typenkonvertierung (6 = 2 + 2 + 2 Punkte)

Welche Zuweisungen sind erlaubt? Fügen Sie in die Platzhalter „erlaubt“ oder „nicht erlaubt“ ein.

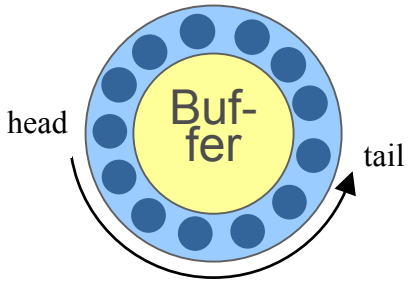
```
public class Conversion
{
  public static void main(String args[])
  {
    char a;
    short b;
    int c;
    float d;
    double e;

    d = a * (float) (b - d) + (int) d;        /* erlaubt */

    b = a + d / (short) e;                    /* nicht erlaubt */

    e = c * 3;                                /* erlaubt */

  }
}
```

3. Arrays und Exceptions (12 = 1 + 2 + 1 + 2 + 1 + 2 + 1 + 2 Punkte)

Die dynamische Datenstruktur `Queue` (FIFO) kann unter zu Hilfename von zyklischen Arrays recht einfach implementiert werden. Vervollständigen Sie im unten angeführten Code die beiden Methoden `put()` und `get()` dahingehend, dass das Verhalten einer Queue folgt und, dass bei Queue-Überlauf eine `QueueOverflowException` und bei Queue-Unterlauf eine `QueueUnderflowException` geworfen wird.

Überlauf: Versuch ein Element hinzuzufügen, obwohl die Queue voll ist.

Unterlauf: Versuch ein Element auszulesen, obwohl die Queue leer ist.




```
class QueueOverflowException extends Exception { }
class QueueUnderflowException extends Exception { }

public class Queue
{
  private int[] queue = null;
  private int head = 0, tail = 0;

  public Queue(int capacity)
  {
    queue = new int[capacity];
  }

  public void put(int value)  throws QueueOverflowException
  {
    if ((tail + 1) % queue.length != head) {
      tail =  (tail + 1) % queue.length ;
      queue[tail] =  value ;
    } else {
      throw new QueueOverflowException() ;
    }
  }

  public int get()  throws QueueUnderflowException
  {
    if (head != tail)
    {
      head =  (head + 1) % queue.length ;
      return  queue[head];
    } else {
      throw new QueueUnderflowException();
    }
  }
}
```

4. Rekursion und Binäre Bäume (12 = 3 + 3 + 3 + 3 Punkte)

Gegeben ist das folgende Gerüst eines binären Baumes. Die Instanzmethode `clone()` soll eine Instanz eines binären Baumes sowie die in diesem Baum gespeicherten Daten rekursiv duplizieren und außerdem den neuen Baum an die aufrufende Methode zurückgeben.

Vervollständigen Sie die Methode `clone()`. Beachten Sie dabei, dass die Methode `clone()` konform der Interfacedefinition `Cloneable` Daten vom Typ `Object` liefert.

```
public class BinTree implements Cloneable
{
  public static BinTree LEER = new BinTree(null, null, null);
  private String data;
  private BinTree left;
  private BinTree right;


  public BinTree(BinTree left, String data, BinTree right)
  {
    this.left = left;
    this.right = right;
    this.data = data;
  }


  public BinTree(String data)
  {
    this(LEER, data, LEER);
  }


  protected Object clone()
  {
    if ( this == LEER )
      return LEER;
    BinTree leftClone = (BinTree) left.clone() ;
    BinTree rightClone = (BinTree) right.clone() ;
    return new BinTree(leftClone, new String(data), rightClone) ;

  }
}
```

5. Zeichen und Stringbuffer (12 = 2 + 2 + 2 + 2 + 2 + 2 Punkte)

Schon zu Zeiten Julius Caesars war Kryptographie ein Thema. So hat dieser einen Algorithmus verwendet, bei dem jedes Zeichen in einer Zeichenkette durch das Zeichen ersetzt wird, welches im Alphabet um `offset` Zeichen weiter hinten liegt. Zeichen, die nicht im Alphabet enthalten sind, werden nicht transformiert. An der Alphabetgrenze wird das Alphabet zyklisch vorgesetzt. Vervollständigen Sie die nachfolgende Klassenmethode so, dass diese einen String nach dieser Methode verschlüsselt.

Beispiel: `offset = 5:` `a` → `f`, `H` → `M`, `y` → `d`, `V` → `A`

```
public static String caesarCrypt(String sentence, int offset)
{
  StringBuffer sb = [ new StringBuffer(sentence); ]

  char actChar;
  char cryptChar;

  for (int i = 0; i < sb.length(); i++) {
    actChar = sb.charAt(i);

    if (actChar >= 'a' && actChar <= 'z') {
      cryptChar = (char) ( [ actChar + offset ] );
      if (cryptChar > 'z')
        cryptChar = (char) (cryptChar [ - ('z' - 'a' + 1) ] );
    } else if (actChar >= 'A' && actChar <= 'Z') {
      cryptChar = (char) ( [ actChar + offset ] );
      if (cryptChar > 'Z')
        cryptChar = (char) (cryptChar [ - ('Z' - 'A' + 1) ] );
    } else {
      cryptChar = actChar;
    }

    sb.setCharAt(i, cryptChar);
  }

  return [ new String(sb) ];
}
```

6. Interfaces und Konstruktoren (14 = 14 * 1 Punkte)

Gegeben ist ein Interface `MeansOfTransportation` – auf Deutsch „Verkehrsmittel“, welches die abstrakte Methode `getNumberOfWheels` zur Ermittlung der Gesamträderzahl eines Verkehrsmittels besitzt.

Implementieren Sie dieses Interface und wenn notwendig, geeignete Konstruktoren für die drei konkreten Verkehrsmittel Car, Tramway und Train.

Implementieren Sie weiters eine Klasse `WheelCounter`, welche die Gesamtzahl der Räder mehrerer Verkehrsmittel ermittelt. Dazu verwenden Sie die Methode `count()`, welche die Anzahl Räder des übergebenen Verkehrsmittels zur Gesamtzahl addiert. Auch soll die Methode `getTotalNumberOfWheels()` existieren, welche die ermittelte Gesamtzahl der Räder liefert.

Car: Hat immer vier Räder.

Tramway: Besteht immer aus zwei Triebwagen (hinten und vorne) mit jeweils sechs Räder und einer bestimmten Anzahl von Wagons mit jeweils vier Rädern. Die Anzahl Wagons soll bei der Erzeugung eines Straßenbahn-Objektes angegeben werden können.

Train: Besteht aus einer bestimmten Anzahl von Lokomotiven mit jeweils acht Rädern und einer bestimmten Anzahl Wagons, wobei jeder Wagon selbst wieder acht Räder besitzt. Sowohl die Lokomotivenzahl als auch die Wagonzahl soll beim Erzeugen eines Zug-Objektes angegeben werden können.

```
public interface MeansOfTransportation
{
  public abstract int getNumberOfWheels();
}



public class WheelCounter
{
  private int totalWheelCount = 0;

  public void count( MeansOfTransportation t )
  {

    totalWheelCount += t.getNumberOfWheels() ;

  }

  public int getTotalNumberOfWheels()
  {

    return totalWheelCount ;

  }
}
```

```
public class Car implements MeansOfTransportation
{
  public int getNumberOfWheels()
  {
    return 4 ;
  }
}
```

```
public class Tramway implements MeansOfTransportation
{
  private int numberOfWagons = 0;

  public Tramway( int numberOfWagons )
  {
    this.numberOfWagons = numberOfWagons ;
  }

  public int getNumberOfWheels()
  {
    return 6 * 2 + 4 * numberOfWagons ;
  }
}
```

```
public class Train implements MeansOfTransportation
{
  private int numberOfLocomotives = 0;
  private int numberOfWagons = 0;

  public Train( int numberOfLocomotives, int numberOfWagons )
  {
    this.numberOfLocomotives = numberOfLocomotives ;
    this.numberOfWagons = numberOfWagons ;
  }

  public int getNumberOfWheels()
  {
    return 8 * (numberOfLocomotives + numberOfWagons) ;
  }
}
```