American University of Central Asia
Software Engineering Department

Programming II (COM 117)
Final Examination

- You have one hour and fifteen minutes to finish the test.
- Circle one or several correct answers.
- In questions with several correct answers you have to select all of them to get a point.
- You can cross answers selected by a mistake.
- You can use the back of the sheets of paper to make notes or to trace code.
- Some *import* statements were removed from the code samples to save space.
- Some *main* entry points were cut short for the same reason.

1. Which of the following are valid Java comments?

   a) *// This will compile*
   b) */** This /// will compile */*
   c) */* /* This will compile */ */*
   d) */* This will compile */*

2. Select the correct statement.

   a) In Java, we instantiate classes from objects.
   b) The object is a blueprint from which we create classes.
   c) The class is a blueprint from which we construct objects.
   d) In Java, classes and objects are interchangeable concepts.

3. Which operator or operators is/are used to create new instances?

   a) Bitwise operators
   b) Logical operators
   c) The *new* operator
   d) The *dot* operator

4. Which operator or operators is/are used to access instance members through a reference variable?

   a) Bitwise operators
   b) Logical operators
   c) The *new* operator
   d) The *dot* operator

5. A variable is shared between all instances of a certain type. What is the most appropriate modifier that you should add to this variable's declaration?

   a) *public*
   b) *int*
   c) *static*
   d) Nothing from all the answers above

6. A variable belongs to an instance of a class. What is the most appropriate modifier that you should add to this variable's declaration?

   a) *public*
   b) *int*
   c) *static*
   d) Nothing from all the answers above

7. Which statements below will not allow to compile the program? Select one answer.

```
class Person {
  static int count = 0;

  private String name;

  Person(String name) {
    this.name = name;

    ++count;
  }

  public String getName() {
    return name;
  }

  public int getCount() {
    return count;
  }
}

public class Main {
  public static void main(...
    Person firstPerson =
      new Person("John");
    Person secondPerson =
      new Person("Jack");

    // 1
    System.out.println(
      Person.count
    );
    // 2
    System.out.println(
      Person.getCount()
    );
    // 3
    System.out.println(
      firstPerson.getCount() +
        "; " +
      secondPerson.getCount()
    );
    // 4
    System.out.println(
      firstPerson.count +
        "; " +
      secondPerson.count
    );
  }
}
```

   a) *1*
   b) *2*
   c) *3*
   d) *4*
   e) *1* and *2*
   f) *1*, *2*, and *4*

8. Which modifier is less restrictive?

   a) *protected*
   b) *package-default*

9. Which modifier is more restrictive—*private* or *package-default*?

   a) *private*
   b) *package-default*
   c) They are the same.

10. What is the problem with the following code?

```
class Student {
  public String name;

  Student(String name) {
    setName(name);
  }

  final void setName(String name) {
    name = name.trim();

    if (name.length() == 0) {
      throw new RuntimeException(
        "Invalid name"
      );
    }

    this.name = name;
  }

  final String getName() {
    return name;
  }
}
```

   a) The class breaks encapsulation rules by making the variable *name* public.
   b) It is not recommended to call the instance method *setName* from the constructor.
   c) It is not a good idea to throw an exception in a setter.
   d) It is better not to do any processing of a parameter in a setter method.

11. Is the *String* class considered mutable or immutable in Java?

   a) The *String* class is mutable in Java.
   b) The *String* class is immutable in Java.
   c) The *String* class is neither mutable nor immutable in Java.
   d) The *String* is a primitive type and not a class in Java.

12. Will the following code print the text with all the whitespace characters replaced with the underscore symbol?

```
public class Main {
  public static void main(...
    String text =
      "The design of the "    +
      "following treatise "   +
      "is to investigate "    +
      "the fundamental laws " +
      "of those operations "  +
      "of the mind by which " +
      "reasoning is performed;";

    text.replace(" ", "_");

    System.out.println(text);
  }
}
```

a) Yes

b) No

c) It will only replace the last whitespace.

13. What will be printed as the last line by the following code?

```
public class Main {
  public static void main(...
    final int Limit = 10_014;

    long startTime;
    startTime = System.nanoTime();
    {
      String result = "";
      for (
        int i = 0, c = 0;
        i < Limit;
        ++i,
        c = (c + 1) % ('z' - 'a' + 1)
      ) {
        result +=
          (char) ('a' + c);
      }

      System.out.println(result);
    }
    long firstEstimatedTime =
      System.nanoTime() - startTime;

    startTime = System.nanoTime();
    {
      StringBuilder result =
        new StringBuilder();

      for (
        int i = 0, c = 0;
        i < Limit;
        ++i,
        c = (c + 1) % ('z' - 'a' + 1)
      ) {
        result.append(
          (char) ('a' + c)
        );
      }

      System.out.println(result);
    }
    long secondEstimatedTime =
      System.nanoTime() - startTime;

    System.out.println(
      firstEstimatedTime >
        secondEstimatedTime
    );
  }
}
```

a) It will print nothing as it will not compile.

b) *true*

c) *false*

14. Is the following *Vector* class considered immutable?

```
class Vector {
  private float x, y;

  Vector(float x, float y) {
    this.x = x;
    this.y = y;
  }

  float getX() {
    return x;
  }

  float getY() {
    return y;
  }

  Vector add(Vector vector) {
    return new Vector(
      x + vector.x,
      y + vector.y
    );
  }
}

public class Main {
  public static void main(...
    Vector firstVector =
      new Vector(1.0f, 2.0f);
    Vector secondVector =
      new Vector(2.0f, 3.0f);
    Vector result =
      firstVector.add(
        secondVector
      );

    System.out.println(
      result.getX() + ", " +
      result.getY()
    );
  }
}
```

a) No, because we can change the value x and y through the constructor.

b) No, because we can access values through getters and thus change them.

c) No, because we can add two vectors together.

d) Yes, the *Vector* class is immutable.

15. The *Exception* in *java.lang* can be classified as...

a) a primitive type

b) a class

c) a checked exception

d) an unchecked exception

16. The *RuntimeException* can be classified as...

a) a primitive type

b) a class

c) a checked exception

d) an unchecked exception

17. What will be the output of the following code?

```
public class Main {
  public static void main(...
    trySomething();
  }

  private static void trySomething() {
    try {
      System.out.print(
        "In try; "
      );

      return;
    } finally {
      System.out.print(
        "In finally; "
      );
    }
  }
}
```

a) *In try;*

b) *In finally;*

c) *In try; In finally;*

d) *In finally; In try;*

18. What will be the output of the following code?

```
public class Main {
  public static void main(...
    trySomething();
  }

  private static void trySomething() {
    try {
      System.out.print(
        "In try; "
      );

      throw new Exception();
    } catch (Exception e) {
      System.out.print(
        "In catch; "
      );

      return;
    } finally {
      System.out.print(
        "In finally; "
      );
    }
  }
}
```

a) *In try;*

b) *In catch;*

c) *In finally;*

d) *In try; In catch;*

e) *In try; In finally;*

f) *In try; In catch; In finally*

19. What will be the output of the following code?

```
public class Main {
  public static void main(...
    try {
      first();
    } catch (Exception e) {
      System.out.println(
        e.getMessage()
      );
    }
  }

  private static void first() {
    second();
  }

  private static void second() {
    try {
      third();
    } catch (Exception ignored) {
      throw new RuntimeException(
        "second"
      );
    }
  }

  private static void third()
      throws Exception {

    throw new Exception(
      "third"
    );
  }
}
```

a) *first*

b) *second*

c) *third*

d) *second third*

e) *third second*

20. What is the full inheritance chain for the *Reptile* class?

```
class Animal {
  //...
}

class Vertebrate extends Animal {
  //...
}

class Invertebrate extends Animal {
  //...
}

class Insect extends Invertebrate {
  //...
}

class Reptile extends Vertebrate {
  //...
}
```

a) *Reptile* > *Object*

b) *Reptile* > *Insect* > *Object*

c) *Reptile* > *Vertebrate* > *Animal*

d) *Object* > *Animal* > *Invertebrate* > *Reptile*

e) *Reptile* > *Vertebrate* > *Animal* > *Object*

f) *Reptile* > *Invertebrate* > *Animal* > *Object*

g) *Reptile* > *Vertebrate* > *Invertebrate* > *Animal* > *Object*

21. Is it possible to use the *new* operator with abstract classes?

    a) Yes, but only if the abstract class does not have abstract methods.

    b) Yes, it is possible even if the class has abstract methods.

    c) Yes, but only for creating an anonymous concrete class from it.

    d) No, it is not possible.

22. An abstract class can contain methods with implementation.

    a) True, it can contain methods with implementation.

    b) True, but only if their visibility modifier is *public*.

    c) True, but only if they are abstract.

    d) False, it can not contain methods with implementation.

23. Is it possible to use the *new* operator with an *interface*?

    a) Yes, but only if the interface is empty.

    b) Yes, it is possible even if the interface is not empty.

    c) Yes, but only for creating an anonymous class from the interface.

    d) No, it is not possible.

24. An interface can contain methods with implementation.

    a) True, it can contain methods with implementation.

    b) True, but only if their visibility modifier is *public*.

    c) True, but only if they are abstract.

    d) False, it can not contain methods with implementation.

25. An interface can serve as a type for a reference variable.

    a) True

    b) False

26. Which of the following variants of the interface declaration will compile and work?

```
// 1
interface Drawable {
  public abstract void draw(Graphics g);
}

// 2
interface Drawable {
  void draw(Graphics g);
}
```

    a) The first one

    b) The second one

    c) Both

    d) None of them

27. How many direct parent classes can a Java class extend?

    a) Zero

    b) Only one

    c) Only two

    d) One or more

28. How many interfaces can a Java class implement?

    a) Zero

    b) Only one

    c) Only two

    d) One or more

29. How many interfaces can a Java interface extend?

    a) Zero

    b) Only one

    c) Only two

    d) One or more

30. What will be the result of trying to compile and run the following code?

```
abstract class ReportBuilder {
  String build() {
    return buildTitle() +
           buildBody()  +
           buildConclusion();
  }

  abstract String buildTitle();
  abstract String buildBody();
  abstract String buildConclusion();
}

class TextReportBuilder
    extends ReportBuilder {

  @Override
  public  String buildTitle() {
    return "Report\n";
  }

  @Override
  public String buildBody() {
    return "Report text.\n";
  }

  @Override
  public String buildConclusion() {
    return "The end.\n";
  }
}

public class Main {
  public static void main(...
    ReportBuilder webReportBuilder =
      new TextReportBuilder() {
        @Override
        public  String buildTitle() {
          return String.format(
            "<h1>%s</h1>",
            super.buildTitle()
                 .trim()
          );
        }

        @Override
        public String buildBody() {
          return String.format(
            "<p>%s</p>",
            super.buildBody()
                 .trim()
          );
        }

        @Override
        public String buildConclusion() {
          return String.format(
            "<p>%s</p>",
            super.buildConclusion()
                 .trim()
          );
        }
      };

    System.out.println(
      webReportBuilder.build()
    );
  }
}
```

    a) It will not compile as the *ReportBuilder* class has methods without a body.

    b) It will not compile as it is not possible to call *super* in such a way.

    c) It will not compile as it is not possible to create a class inline without a name.

    d) The code does not make any sense and will not compile.

    e) It will compile, run, and print the report in a custom format of the *webReportBuilder*.

    f) It will compile, run, and print just the plain text from the *TextReportBuilder*.

31. What will be the result of trying to compile and run the following code?

```
import java.util.Scanner;

class Node<T> {
  private Node<T> next;
  private T data;

  Node() {
    this(null, null);
  }

  Node(T data) {
    this(null, data);
  }

  Node(Node<T> next, T data) {
    this.next = next;
    this.data = data;
  }

  public Node<T> getNext() {
    return next;
  }

  public void setNext(Node<T> next) {
    this.next = next;
  }

  public T getData() {
    return data;
  }

  public void setData(T data) {
    this.data = data;
  }
}

public class Main {
  public static void main(...
    Scanner scanner =
      new Scanner("1 2 3 4 5 6 7");
    Node<int> listRoot =
      readNumbers(scanner);

    printNumbers(listRoot);
  }

  public static Node<int> readNumbers(
                    Scanner scanner
                  ) {
    Node<int> root = null;

    if (scanner.hasNextInt()) {
      root =
        new Node<int>(
          scanner.nextInt()
        );
    }

    Node<int> current = root;
    while (scanner.hasNextInt()) {
      current.setNext(
        new Node<int>(
          scanner.nextInt()
        )
      );
      current =
        current.getNext();
    }

    return root;
  }

  public static void printNumbers(
                      Node<int> node
                    ) {
    while (node != null) {
      System.out.print(
        node.getData() + " "
      );
      node =
        node.getNext();
    }
  }
}
```

    a) It will compile and print the sequence *1 2 3 4 5 6 7*

    b) It will not compile because the primitive type *int* can not be used as a type parameter. *int* can be autoboxed by Java into *Integer*, but cannot be used as a type parameter.

    c) It will not compile because the class *Node* cannot aggregate (or use a *has-a* relationship) with a variable *next* of the same type *Node*.

    d) It will not compile because there are no types with the name *T* in the Java standard library.