American University of Central Asia
Software Engineering Department

Programming II (COM 117)
Midterm 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. Describe the relationship between an object and a class in Java.

   a) A class is an interchangeable term for an object.
   b) A class is a template from which we can build objects.
   c) An object is a recipe from which we can instantiate classes.

3. Instance variables in Java are used to...

   a) represent the state of an object.
   b) represent the object's behavior.
   c) represent the object's identity.

4. Instance methods in Java are used to...

   a) represent the state of an object.
   b) represent the object's behavior.
   c) represent the object's identity.

5. According to the Java naming conventions, which of the following can be used as a class name?

   a) *dog*
   b) *Expression_Parser*
   c) *GAME_ENTITY*
   d) *JButton*

6. What is the most appropriate accessor's name for a string variable named *definition* in Java?

   a) *definition*
   b) *definition!*
   c) *getDefinition*
   d) *setDefinition*

7. What is the most appropriate mutator's name for a boolean variable named *active* in Java?

   a) *setIsActive*
   b) *setActive*
   c) *active*
   d) *active?*
   e) *isActive*

8. The name of the boolean variable is *enabled*. What is the most appropriate name for its accessor method?

   a) *setEnabled*
   b) *getEnabled*
   c) *enabled?*
   d) *isEnabled*

9. What is the sequence of tasks the *new* operator should perform in Java in order to instantiate a new object?

   a) Call the constructor, return a reference to the new object.
   b) Allocate space on a stack, call the constructor, return a reference to the new object.
   c) Allocate space on a heap, call the constructor, return a reference to the new object.

10. Is it possible to have more than one constructor in Java?

    a) Yes
    b) No

11. Is it possible in Java not to declare a constructor at all?

    a) Yes
    b) No

12. The constructor declared in sources without any parameters is usually called...

    a) a no-argument constructor.
    b) a parameterized constructor.
    c) a null-constructor.

13. Does Java require to have a parameterized constructor?

    a) Yes
    b) No

14. True or false: a no-argument constructor is an interchangeable term for *a default constructor*?

    a) True
    b) False

15. A chaining call with *this* to another constructor must be the first statement in a constructor. True or false?

    a) True
    b) False

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

```
public static void main(...
  Color[] color =
    new Color[10];

  System.out.println(
    color[0] == null
  );
}
```

    a) *null*
    b) *true*
    c) *false*
    d) The program is incorrect and will not compile. The elements in the array are not initialized.

17. The instance of *Color* in the following example will be stored by Java...

```
public static void main(...
  Color fillColor =
    new Color(255, 0, 0);
}
```

    a) on a heap.
    b) on a stack.

18. The reference to the object of type *Color* in the following example will be stored...

```
public static void main(...
  Color fillColor =
    new Color(255, 0, 0);
}
```

    a) on a heap.
    b) on a stack.

19. What is the correct sequence of visibility modifiers placed from the most to the least restrictive ones?

a) package-default, private, protected, public

b) private, package-default, protected, public

c) private, protected, package-default, public

d) private, package-default, public, protected

20. Can constructors be made private?

a) Yes, constructors can be made private.

b) No, constructors can not be marked as private.

c) Only parameterized constructors can be made private.

21. What is the output of the following program?

```
public class Point {
  private static int x, y;

  public Point(int newX, int newY) {
    x = newX;
    y = newY;
  }

  public int getX() { return x; }
  public int getY() { return y; }

  public static void main(...
    Point firstPoint =
      new Point(1, 2);
    Point secondPoint =
      new Point(3, 4);
    Point thirdPoint =
      new Point(5, 6);

    System.out.printf(
      "%d, %d; ",
      firstPoint.getX(),
      firstPoint.getY()
    );
    System.out.printf(
      "%d, %d; ",
      secondPoint.getX(),
      secondPoint.getY()
    );
    // Yes, the secondPoint
    // once again.
    // It is not a copy-paste
    // error.
    System.out.printf(
      "%d, %d",
      secondPoint.getX(),
      secondPoint.getY()
    );
  }
}
```

a) 1, 2; 3, 4; 5, 6

b) 1, 2; 3, 4; 3, 4

c) 3, 4; 3, 4; 3, 4

d) 5, 6; 5, 6; 5, 6

22. Is it possible to compile the following program?

```
public class Flower {
  private String type =
    "Daffodil";

  public static void printType() {
    System.out.println(
      type
    );
  }
}
```

...if the class is used in the following way

```
public static void main(...
  Flower flower =
    new Flower();

  flower.printType();
}
```

a) Yes, we can compile and run the code.

b) No, as we do not have a constructor in the class *Flower*.

c) No, as it is not possible to initialize an instance variable in that way.

d) No, as it is not possible to access an instance field from a static method.

23. What are the problems with the following code defined in a file named *Pet.java*?

```
class Cat {
  public String name;

  cat(String name) {
    name = name;
  }

  String getName() {
    return name;
  }
}

public class Pet {
  public static void main(...
    Cat cat =
      new Cat("Bobik");

    System.out.println(
      cat.getName()
    );
  }
}
```

a) The constructor name is not the same as the name of the class.

b) The *name* variable representing the parameter to the constructor is assigned to itself.

c) The class name *Cat* and the file name *Pet* are not the same.

d) The class *Cat* breaks encapsulation rules by making the instance variable *name* public.

e) The constructor and the getter should be made public.

f) The cat's name is *Bobik*.

24. Which of the following types can be considered immutable?

```
// A
import java.util.Date;

class Book {
  int id; String name; Date due;

  Book(
    int id,
    String name,
    Date due
  ) {
    this.id = id;
    this.name = name;
    this.due = due;
  }

  int getId() {
    return id;
  }

  String getName() {
    return name;
  }

  Date getDue() {
    return due;
  }
}
```

or

```
// B
import java.util.Date;

class Book {
  int id; String name; Date due;

  Book(
    int id,
    String name,
    Date due
  ) {
    this.id = id;
    this.name = name;
    this.due = (Date) due.clone();
  }

  int getId() {
    return id;
  }

  String getName() {
    return name;
  }

  Date getDue() {
    return (Date) due.clone();
  }
}
```

a) *A*

b) *B*

c) None

25. In a certain programming language it is possible to have two base classes for a class. What is the name of the inheritance model used by this language?

a) Composition

b) Single inheritance

c) Multiple inheritance

26. Consider the following Java program. What should be the result of trying to compile and run it?

```
class Kernel {
  private String name;

  public Kernel() {
    this("Unknown Kernel");
  }

  public Kernel(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }
}

class BSD extends Kernel {
  public BSD() {
    super("BSD Kernel");
  }
}

class Mach extends Kernel {
  public Mach() {
    super("Mach Kernel");
  }
}

class XNU extends BSD, Mach { }

public class OS {
  public static void main(...
    XNU xnuKernel =
      new XNU();

    System.out.println(
      xnuKernel.getName()
    );
  }
}
```

a) *Unknown Kernel*

b) *BSD Kernel*

c) *Mach Kernel*

d) *BSD Kernel Mach Kernel*

e) *Unknown Kernel BSD Kernel Mach Kernel*

f) It will not compile because Java does not support the multiple inheritance model.

g) Java does allow multiple inheritance, but the program will not compile because the compiler will get confused which *getName()* method from its parent classes to call.

27. What are the requirements for method overriding in Java?

a) The signature of a method in a derived class must be the same as the signature in the base class.

b) The return type of a method in a derived class must be the same as the return type in the base class.

c) The method in the derived class MUST call the method in the superclass with *super* at the beginning of the method.

28. Is it possible to have multiple signatures in method overloading in Java?

a) No, and the return type must be the same.

b) No, but the return type can be different.

c) Yes, that is the whole point of method overloading.

29. Multiple implementations of the same method in parent and child classes force Java to use...

a) static binding.

b) dynamic binding.

c) the *instanceof* operator.

30. A call to a constructor of a superclass with *super* must be the first statement in a constructor. True or false?

a) True

b) False

31. Is it considered a good practice to call parent methods with *super*?

a) Yes

b) No

32. What is the full inheritance chain for the *Human* class.

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

class Reptile extends Animal {
  //...
}

class Mammal extends Animal {
  //...
}

class Human extends Mammal {
  //...
}
```

a) *Human* > *Object*

b) *Human* > *Mammal* > *Animal*

c) *Human* > *Mammal* > *Animal* > *Object*

d) *Human* > *Mammal* > *Reptile* > *Animal*

e) *Human* > *Mammal* > *Reptile* > *Animal* > *Object*

33. What will be the result of running the following code?

```
class Dog {
  public String bark() {
    return "Arf!";
  }

  // ...
}

class GermanShepherd extends Dog {
  public String bark() {
    return "Wuff!";
  }
}

class SiberianHusky extends Dog {
  public String bark() {
    return "Gav!";
  }
}

public class Main {
  public static void main(...
    Dog firstDog =
      new Dog();
    Dog secondDog =
      new GermanShepherd();
    Dog thirdDog =
      new SiberianHusky();
    GermanShepherd fourthDog =
      new GermanShepherd();
    SiberianHusky fifthDog =
      new SiberianHusky();

    System.out.println(
      firstDog.bark()  + " " +
      secondDog.bark() + " " +
      thirdDog.bark()  + " " +
      fourthDog.bark() + " " +
      fifthDog.bark()
    );
  }
}
```

a) *Arf! Wuff! Gav! Wuff! Gav!*

b) *Arf! Arf! Arf! Wuff! Gav!*

c) *Arf! Arf! Wuff! Arf! Gav! Wuff! Gav!*

d) *Arf! Arf! Wuff! Arf! Gav! Arf! Wuff! Arf! Gav!*

34. Assume we are using the classes *Dog*, *GermanShepherd*, and *SiberianHusky* declared in the previous question. What could be the result of trying to compile and run the following code?

```
public class Main {
  public static void main(...
    GermanShepherd dog =
      new Dog();

    System.out.println(
      dog.bark()
    );
  }
}
```

a) It will compile and print *Arf!*

b) It will compile and print *Wuff!*

c) It will compile and print *Gav!*

d) We can not compile this code without errors.

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

```
class Person {
  protected String name;

  public Person(String newName) {
    name = newName;
  }

  public String getName() {
    return name;
  }
}

class Teacher extends Person {
  public String getName() {
    return "T:" + name;
  }
}

class Student extends Person {
  public String getName() {
    return "S:" + super.getName();
  }
}

public class Main {
  public static void main(...
    Person person =
      new Person("John");
    Teacher teacher =
      new Teacher("Margaret");
    Student student =
      new Student("Jack");

    printInformation(person);
    printInformation(teacher);
    printInformation(student);
  }

  private static void printInformation(
                        Person person
                      ) {
    System.out.print(
      person.getName() + " "
    );
  }
}
```

a) *John Margaret Jack*

b) *John T:Margaret Jack*

c) *John Margaret S:Jack*

d) *John T:Margaret S:Jack*

e) It will not compile as we are trying to read the *name* field directly from the base class without a getter in *Teacher*.

f) It will not compiler as we are trying to call a parent method with *super* from *Student* which can only work from inside a constructor.

g) It will not compiler as we do not have a no-argument constructor in the superclass.

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

```
import java.util.Scanner;

class Node {
  private Node next;
  private Object data;

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

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

  Node(Node next, Object data) {
    this.next = next;
    this.data = data;
  }

  public Node getNext() {
    return next;
  }

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

  public Object getData() {
    return data;
  }

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

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

    printNumbers(listRoot);
  }

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

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

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

    return root;
  }

  public static void printNumbers(
                       Node 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* is not a class and does not inherit from *Object*.

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

d) It will not compile as the code above does not make any sense.