Name _____

CISC 3115 – Modern Programming Techniques
Spring 2026
Exam 1

Part I. Answer all questions in the space to the left. (60 points)

_____ 1) If a balance is changed directly instead of using the withdrawal operation, the program:
a) bypasses rule logic
b) improves encapsulation
c) reduces memory usage
d) guarantees correctness

_____ 2) In a parallel-array design, the array index represents:
a) the record identity
b) the array length
c) the memory location
d) the sorting order

_____ 3) An operations class typically:
a) contains behavior but little or no entity data
b) stores primitive variables only
c) contains no methods
d) represents compiler rules

_____ 4) If a method changes a field of an object passed to it:
a) the object keeps the change
b) the change disappears
c) the compiler rejects it
d) Java copies the object automatically

_____ 5) Instance variables are used to:
a) store attributes of an object
b) store loop counters
c) store compiler instructions
d) store array indices

_____ 6) Instance methods are methods that:
a) operate on specific objects
b) run only once
c) belong to arrays
d) cannot access variables

_____ 7) In `account.deposit(500)`, the object before the dot is the:

a) receiver
b) parameter
c) argument
d) return value

_____ 8) The statement `BankAccount[] accounts = new BankAccount[100];` creates:

a) an array capable of storing references to `BankAccount` objects
b) 100 `BankAccount` objects automatically
c) a primitive array
d) a loop

_____ 9) If an object reference variable is copied to another variable:

a) both variables refer to the same object
b) Java creates a new object
c) the original object is deleted
d) only primitive values copy

_____ 10) Encapsulation helps ensure that:

a) rules governing an entity are enforced through methods
b) arrays are unnecessary
c) programs run faster
d) loops are removed

_____ 11) Changing object state normally occurs through:

a) method calls
b) comments
c) array declarations
d) compiler directives

_____ 12) Objects help organize programs by:

a) grouping state and behavior together
b) removing variables
c) eliminating arrays
d) avoiding methods

_____ 13) An instance of a class is:

a) an object created from that class
b) a loop
c) a compiler rule
d) a variable name

_____ 14) The state of an object is stored in its:
a) methods
b) instance variables
c) constructors
d) loops

_____ 15) A constructor is primarily used to:
a) initialize the object
b) declare new classes
c) terminate execution
d) create arrays

_____ 16) Constructors differ from ordinary methods because they:
a) cannot take parameters
b) cannot access fields
c) have no return type
d) must be static

_____ 17) Constructor parameters typically provide:
a) access modifiers
b) loop counters
c) initial values for instance variables
d) names of classes

_____ 18) Instance variables are usually declared:
a) `public`
b) `private`
c) `static`
d) temporary

_____ 19) Methods that form the external interface of a class are typically:
a) `static`
b) `private`
c) `final`
d) `public`

_____ 20) Delegating from one constructor to another helps:
a) remove instance variables
b) replace methods
c) avoid duplicate initialization logic
d) disable encapsulation

_____ 21) The syntax `this(...)` inside a constructor:
a) declares a variable
b) calls a superclass method
c) creates a new object
d) calls another constructor in the same class

_____ 22) A constructor that takes no arguments is called:
a) a default constructor
b) a message constructor
c) a primitive constructor
d) a recursive constructor

_____ 23) If a programmer writes any constructor:
a) objects cannot be created
b) all constructors become public
c) fields become static
d) the implicit default constructor is not generated

_____ 24) Shadowing occurs when:
a) two classes share the same name
b) a parameter has the same name as an instance variable
c) a constructor calls another
d) two objects share memory

_____ 25) The keyword `this` inside a constructor refers to:
a) the class name
b) the compiler
c) a local variable
d) the current object

_____ 26) A class should avoid a default constructor when:
a) it has many methods
b) valid objects require caller input
c) it uses Strings
d) fields are private

_____ 27) Object creation is best understood as:
a) new allocates memory then the constructor initializes the object
b) constructors run before objects exist
c) methods create objects
d) objects initialize themselves

_____ 28) If a constructor parameter shadows an instance variable, using this allows:
a) automatic recursion
b) removal of variables
c) creation of new parameters
d) access to the instance variable

_____ 29) Encapsulation is violated when:
a) fields are private
b) constructors initialize fields
c) methods validate input
d) external code directly modifies instance variables

_____ 30) Composition in a class design means:
a) one class contains objects of another class as instance variables
b) methods are stored outside the class
c) a class inherits from two parent classes
d) objects cannot contain other objects

_____ 31) When a containing object is created, the contained objects are usually:
a) initialized inside the constructor
b) stored in static variables
c) shared by all objects
d) created by the compiler automatically

_____ 32) When an object is printed using `System.out.println(obj)`, Java:
a) compares references
b) automatically calls `obj.toString()`
c) creates a copy constructor
d) calls equals

_____ 33) If a class contains other objects, its `toString` method often:
a) calls `toString` on the contained objects
b) creates new contained objects
c) compares contained objects
d) ignores the contained objects

_____ 34) Semantic equality between objects means:
a) objects are printed the same way
b) objects have the same constructor
c) objects occupy the same memory location
d) objects represent the same logical value

_____ 35) Using == with objects tests:
a) whether two references refer to the same object
b) whether objects have equal data
c) whether constructors are identical
d) whether objects have equal `toString` values

_____ 36) If two variables alias the same object and one modifies it:
a) the second variable becomes null
b) the change is visible through both variables
c) a new object is created
d) the program cannot compile

_____ 37) The primary purpose of a copy constructor is to:
a) replace equals
b) call `toString` automatically
c) avoid aliasing problems
d) initialize loops

_____ 38) A class implementing `equals` typically compares:
a) its comments
b) its constructor names
c) its loop counters
d) its instance variables

_____ 39) Aliasing can lead to bugs because:
a) constructors fail
b) objects cannot be created
c) multiple references modify the same object
d) arrays disappear

_____ 40) Composition helps program design because:
a) arrays are removed
b) complex objects can be built from simpler objects
c) loops disappear
d) inheritance is eliminated

_____ 41) Chaining `toString` through composition means:
a) removing contained objects
b) creating nested constructors
c) sharing references
d) calling `toString` on contained objects

_____ 42) A copy constructor helps maintain:
a) aliasing
b) method overloading
c) static state
d) object independence

_____ 43) Which form is normally used when calling a static method?
```
a) object.method()
b) ClassName.method()
c) method(object)
d) new.method()
```

_____ 44) Why can a static method not directly access instance variables?
a) instance variables require inheritance
b) instance variables belong to specific objects
c) instance variables are private
d) instance variables must be final

_____ 45) Why must the main method be static?
a) the JVM invokes it before any objects exist
b) it must access all fields
c) it cannot return a value
d) it must create arrays

_____ 46) If a read operation cannot construct a valid object, what is commonly returned?
a) false
b) zero
c) null
d) an empty string

_____ 47) Responsibility-driven programming emphasizes that:
a) objects manage their own behavior
b) arrays control program flow
c) loops define state
d) applications modify all fields

_____ 48) Utility classes most often contain:
a) static operations
b) instance arrays
c) recursive methods only
d) container objects

_____ 49) Why might a program use a container class rather than directly exposing an array?
a) to enforce rules on how elements are managed
b) to remove object creation
c) to avoid constructors
d) to eliminate references

_____ 50) In an object container, adding a new element typically increases:
a) array capacity
b) logical size
c) class size
d) method count

_____ 51) Which statement about static variables is correct?
a) they exist once per object
b) they exist once per class
c) they are always private
d) they cannot store primitives

_____ 52) Why might immutable objects reduce certain programming errors?
a) their state cannot be changed unexpectedly
b) they eliminate constructors
c) they avoid references
d) they remove arrays

_____ 53) Why are read methods often static?
a) they construct objects without needing an existing instance
b) they must modify fields
c) they return primitives
d) they call constructors automatically

_____ 54) Why are constants such as `BLACK` declared `static final`?
a) to allow modification
b) to reduce primitive memory
c) to share one constant instance
d) to enable polymorphism

_____ 55) Why implement contains using find in the container class?
a) reuse search logic
b) enable sorting
c) prevent recursion
d) increase complexity

_____ 56) What distinguishes `MutableInt` from `ImmutableInt`?

a) static variables
b) inheritance
c) method overloading
d) internal state modification

_____ 57) Why do `MutableInt` methods return `this`?

a) enable chaining
b) enforce immutability
c) avoid allocation
d) simplify constructors

_____ 58) Which situation best illustrates immutability?

a) a method modifies an object's field
b) a method returns a new object
c) a method deletes a variable
d) a method uses recursion

_____ 59) Which expression demonstrates chaining in `MutableInt`?

```
a) i0.reset()
b) new MutableInt()
c) i0.increaseBy(i1)
d) i0.increaseBy(i1).multiplyBy(i2).divideBy(i3)
```

_____ 60) Which task demonstrates composition in the `Quadrilateral` class?

a) using `Point` objects
b) storing primitive `ints`
c) declaring static fields
d) calling recursion

Part II. Answer all questions (40 points)

_____ 61) Given the following `Student` class that contains a `Name` instance variable:

```
 1   class Student {
 2         Student(Name name) {
 3               id = nextId;
 4               nextId++;
 5               this.name = name;
 6         }
 7         …
 8         …
 9         int id;
10         Name name;
11         static int nextId = 1001;
12   }
```

a) Why is `nextId` static? What is the purpose of the `nextId` variable? What is happening in lines `3` and `4`?

`nextId` is used to assign sequential id's to `Student` objects as they are created. We want this number to be unique so we make it static (i.e., a class variable) so there is only one copy of it in the class. In line3 3 and 4 we assign the current value of `nextId` to the `id` instance variable (of which there is one per `Student` instance/object, and then update `nextId` so the next created object gets a different (unique and sequential) value in its `id`.

b) Assuming `Name` has a `read` method, write the `read` method for this class. (Remember, read methods accept a `Scanner` and return an object of their defining class created from data read in). Make sure you handle the situation where there is no more data.

```
public static Student read(Scanner scanner) {
     Name name = Name.read(scanner);
     if (name == null) return null;
     return new Student(name);
}


public static Name read(Scanner scanner) {
     if (!scanner.hasNext()) return null;
     String last = scanner.next();
     String first = scanner.next();
     return new Name(last, first);
}
```

Makes no difference as to which order you read last and first name

_____ 62) Given the following app (the `main` is in the `CarDemo` class):

```
class Engine {
     public Engine(boolean isHybrid, int numCylinders) {
          this.isHybrid = isHybrid;
          this.numCylinders = numCylinders;
     }
     public Engine() {this(false, 4);}

     public int mpg() {return 60/numCylinders;}

     public String toString() {
          return numCylinders + " cylinder" + (isHybrid ? " hybrid" : "");
     }

     private boolean isHybrid;
     private int numCylinders;
}
```

```
class Car {
     public Car(int year, String model,  Engine engine) {
          this.year = year;
          this.model = model;
          this.engine = engine;
     }
     public Car(String model) {this(2018, model, new Engine());}

     public int mpg() {return engine.mpg();}

     public String toString() {
          return "a " + year + " " + model + " " + engine;
     }

     private int year;
     private String model;
     private Engine engine;
}
```

```
class CarDemo {
     public static void main(String [] args) {
          Engine engine = new Engine(true, 6);
          Car car1 = new Car(2017, "Impala", engine);
          System.out.println("car1: " + car1);
          System.out.println("car1's mpg: " + car1.mpg());

          Car car2 = new Car("Malibu");
          System.out.println("car2: " + car2);
          System.out.println("car2's mpg: " + car2.mpg());
     }
}
```

a.(**7 points**)What is output by the program's execution?

```
car1: a 2017 Impala 6 cylinder hybrid
car1's mpg: 10 mpg
car2: a 2018 Malibu 4 cylinder
car2's mpg: 15
```

b. (**3 points**) Which method is a *delegation method*? What makes it a delegation method?

The `mpg` method of the `Car` class … all it does is call the `mpg` method of `Engine`

c. Optional (**3 points**): Can you see a problem with an `Engine` being created with zero (0) cylinders (it's more than simply – *'a Car can't have 0 cylinders'*)? What is the problem, and what would you do about it (nothing detailed—just a sentence is fine)?

`mpg` divides 60 by the number of cylinders… for 0 cylinders this would result in a 0-divide

_____ 63)

a) (**15 points**) Write a class named `Integer` with the following:

- an `int` instance variable
- 3 constructors:
  - a 1-arg constructor accepting an int,
  - a default constructor that initializes the instance variable to 0,
  - a copy constructor.

  For full credit, have the default constructor delegate to the 1-arg constructor
- a `toString` method that returns the value of the instance variable
- an `equals` method
- an `add` method that accepts another `Integer` and returns a new object corresponding to the sum of the receiver and the parameter (i.e., an immutable addition operation)
- an `increaseBy` method that accepts another `Integer`, performs an *in-place* add on the receiver and returns (a reference to) the receiver (i.e., a mutable addition operation)

```
class Integer {
     Integer(int val) {this.val = val;}
     Integer() {this(0);}
     Integer(Integer other) {this.val = other.val;}

     Integer add(Integer other) {
         return new Integer(this.val + other.val);
     }
     Integer increaseBy(Integer other) {
         this.val += other.val;
         return this;
     }

     public boolean equals(Integer other) {
         return this.val == other.val;
     }
     public String toString() {return val+"";}

     private int val;
}
```

b) (**7 points**) Write a fragment of code that illustrates usage of the above class (i.e., declares variables, illustrates the various constructors and methods). Include an example of chaining.

```
class IntegerApp {
    public static void main(String [] args) {
        Integer
            i1 = new Integer(12),
            i2 = new Integer(),
            i3 = i1,
            i4 = new Integer(3);

        System.out.println("i1: " + i1 + " i2: " + i2 +
            " i3: " + i3 + " i4: " + i4);

        System.out.println("i1.equals(i1): " + i1.equals(i1));
        System.out.println("i1.equals(i2): " + i1.equals(i2));
        System.out.println("i1.equals(i3): " + i1.equals(i3));
        System.out.println("i1.equals(i4): " + i1.equals(i4));

        System.out.println("i1.add(i4): " + i1.add(i4));
        System.out.println("i1.increaseBy(i4): " + i1.increaseBy(i4));
        System.out.println("i1: " + i1);
        }
}
```