Full Name: _____

Student ID#: _____

CSE 1325 OBJECT-ORIENTED PROGRAMMING

Exam #1 «---» R 1||2||3||4 1||2||3||4||5||6... 1||2||3 003||002 «---» Exam #1

Instructions

1. Students are allowed pencils, erasers, UTA Student ID, and beverage only. **A UTA Student ID is required to turn in the exam.**
2. All books, bags, backpacks, phones, **smart watches**, **ear buds**, and other electronics, etc. must be placed along the walls. **Silence all notifications.**
3. PRINT your name and student ID at the top of this page **and every additional pastel coding sheet**, and verify that you have all 10 pages.
4. **Read every question completely before you start to answer it.** If you have a question, please raise your hand. You may or may not get an answer, but it won't hurt to ask.
5. If you leave the room, you may not return.
6. You are required to SIGN and ABIDE BY the following Honor Pledge for each exam this semester.

Honor Pledge

On my honor, I pledge that I will not attempt to communicate with another student, view another student's work, or view any unauthorized notes or electronic devices during this exam. I understand that the professor and the CSE 1325 Course Curriculum Committee have zero tolerance for cheating of any kind, and that any violation of this pledge or the University honor code will result in an automatic grade of zero for the semester and referral to the Office of Student Conduct for scholastic dishonesty.

Student Signature: _____

Vocabulary

Write the word or phrase from the Words list below to the left of the definition that it best matches. Each word or phrase is used at most once, but some will not be used. {10 at 2 points each}

Vocabulary

Word	Definition
1	A class that cannot be instantiated
2	A method declared with no implementation
3	Specifying a general interface while hiding implementation details
4	A procedure for solving a specific problem, expressed in terms of an ordered set of actions to execute
5	An expression that, if false, indicates a program error

Word List

Abstract Class	Abstract Method	Abstraction	Algorithm	Assertion
Class	Constructor	Data Validation	Declaration	Definition
Destructor	Encapsulation	Enumerated Type	Exception	Field
Garbage Collector	Getter	Inheritance	Interface	Method
Multiple Inheritance	Namespace	Object	Object-Oriented Programming	Operator
Override	Package	Primitive type	Setter	Subclass
Superclass	UML	Validation Rules	Variable	Version Control

Vocabulary

Word	Definition
1 Abstract Class	A class that cannot be instantiated
2 Abstract Method	A method declared with no implementation
3 Abstraction	Specifying a general interface while hiding implementation details
4 Algorithm	A procedure for solving a specific problem, expressed in terms of an ordered set of actions to execute
5 Assertion	An expression that, if false, indicates a program error

Vocabulary

Word	Definition
6	A template encapsulating data and code that manipulates it
7	A special class member that creates and initializes an object from the class
8	Ensuring that a program operates on clean, correct, and useful data
9	A statement that introduces a name with an associated type into a scope
10	A declaration that also fully specifies the entity declared

Word List

Abstract Class	Abstract Method	Abstraction	Algorithm	Assertion
Class	Constructor	Data Validation	Declaration	Definition
Destructor	Encapsulation	Enumerated Type	Exception	Field
Garbage Collector	Getter	Inheritance	Interface	Method
Multiple Inheritance	Namespace	Object	Object-Oriented Programming	Operator
Override	Package	Primitive type	Setter	Subclass
Superclass	UML	Validation Rules	Variable	Version Control

Vocabulary

Word	Definition
6 Class	A template encapsulating data and code that manipulates it
7 Constructor	A special class member that creates and initializes an object from the class
8 Data Validation	Ensuring that a program operates on clean, correct, and useful data
9 Declaration	A statement that introduces a name with an associated type into a scope
10 Definition	A declaration that also fully specifies the entity declared

Vocabulary

Word	Definition
11	A special class member that cleans up when an object is deleted
12	Bundling data and code into a restricted container
13	A data type that includes a fixed set of constant values called enumerators
14	An object created to represent an error or other unusual occurrence and then propagated via special mechanisms until caught by special handling code
15	A class member variable

Word List

Abstract Class	Abstract Method	Abstraction	Algorithm	Assertion
Class	Constructor	Data Validation	Declaration	Definition
Destructor	Encapsulation	Enumerated Type	Exception	Field
Garbage Collector	Getter	Inheritance	Interface	Method
Multiple Inheritance	Namespace	Object	Object-Oriented Programming	Operator
Override	Package	Primitive type	Setter	Subclass
Superclass	UML	Validation Rules	Variable	Version Control

Vocabulary

Word	Definition
11 Destructor	A special class member that cleans up when an object is deleted
12 Encapsulation	Bundling data and code into a restricted container
13 Enumerated Type	A data type that includes a fixed set of constant values called enumerators
14 Exception	An object created to represent an error or other unusual occurrence and then propagated via special mechanisms until caught by special handling code
15 Field	A class member variable

Vocabulary

Word	Definition
16	A program that runs in managed memory systems to free unreferenced memory
17	A method that returns the value of a private variable
18	Reuse and extension of fields and method implementations from another class
19	A reference type containing only method signatures, default methods, static methods, constants, and nested types
20	A function that manipulates data in a class

Word List

Abstract Class	Abstract Method	Abstraction	Algorithm	Assertion
Class	Constructor	Data Validation	Declaration	Definition
Destructor	Encapsulation	Enumerated Type	Exception	Field
Garbage Collector	Getter	Inheritance	Interface	Method
Multiple Inheritance	Namespace	Object	Object-Oriented Programming	Operator
Override	Package	Primitive type	Setter	Subclass
Superclass	UML	Validation Rules	Variable	Version Control

Vocabulary

Word	Definition
16 Garbage Collector	A program that runs in managed memory systems to free unreferenced memory
17 Getter	A method that returns the value of a private variable
18 Inheritance	Reuse and extension of fields and method implementations from another class
19 Interface	A reference type containing only method signatures, default methods, static methods, constants, and nested types
20 Method	A function that manipulates data in a class

Vocabulary

Word	Definition
21	A subclass inheriting class members from two or more superclasses
22	A named scope
23	An instance of a class containing a set of encapsulated data and associated methods
24	A style of programming focused on the use of classes and class hierarchies
25	A short string representing a mathematical, logical, or machine control action

Word List

Abstract Class	Abstract Method	Abstraction	Algorithm	Assertion
Class	Constructor	Data Validation	Declaration	Definition
Destructor	Encapsulation	Enumerated Type	Exception	Field
Garbage Collector	Getter	Inheritance	Interface	Method
Multiple Inheritance	Namespace	Object	Object-Oriented Programming	Operator
Override	Package	Primitive type	Setter	Subclass
Superclass	UML	Validation Rules	Variable	Version Control

Vocabulary

Word	Definition
21 Multiple Inheritance	A subclass inheriting class members from two or more superclasses
22 Namespace	A named scope
23 Object	An instance of a class containing a set of encapsulated data and associated methods
24 Object-Oriented Programming	A style of programming focused on the use of classes and class hierarchies
25 Operator	A short string representing a mathematical, logical, or machine control action

Vocabulary

Word	Definition
26	A subclass replacing its superclass' implementation of a method
27	A grouping of related types providing access protection and namespace management
28	A data type that can typically be handled directly by the underlying hardware
29	A method that changes the value of a private variable
30	The class inheriting members

Word List

Abstract Class	Abstract Method	Abstraction	Algorithm	Assertion
Class	Constructor	Data Validation	Declaration	Definition
Destructor	Encapsulation	Enumerated Type	Exception	Field
Garbage Collector	Getter	Inheritance	Interface	Method
Multiple Inheritance	Namespace	Object	Object-Oriented Programming	Operator
Override	Package	Primitive type	Setter	Subclass
Superclass	UML	Validation Rules	Variable	Version Control

Vocabulary

Word	Definition
26 Override	A subclass replacing its superclass' implementation of a method
27 Package	A grouping of related types providing access protection and namespace management
28 Primitive type	A data type that can typically be handled directly by the underlying hardware
29 Setter	A method that changes the value of a private variable
30 Subclass	The class inheriting members

Vocabulary

Word	Definition
31	The class from which members are inherited
32	The standard visual modeling language used to describe, specify, design, and document the structure and behavior of object-oriented systems
33	Algorithmically enforceable constraints on the correctness, meaningfulness, and security of input data
34	A block of memory associated with a symbolic name that contains a primitive data value or the address of an object instance
35	The task of keeping a system consisting of many versions well organized

Word List

Abstract Class	Abstract Method	Abstraction	Algorithm	Assertion
Class	Constructor	Data Validation	Declaration	Definition
Destructor	Encapsulation	Enumerated Type	Exception	Field
Garbage Collector	Getter	Inheritance	Interface	Method
Multiple Inheritance	Namespace	Object	Object-Oriented Programming	Operator
Override	Package	Primitive type	Setter	Subclass
Superclass	UML	Validation Rules	Variable	Version Control

Vocabulary

Word	Definition
31 Superclass	The class from which members are inherited
32 UML	The standard visual modeling language used to describe, specify, design, and document the structure and behavior of object-oriented systems
33 Validation Rules	Algorithmically enforceable constraints on the correctness, meaningfulness, and security of input data
34 Variable	A block of memory associated with a symbolic name that contains a primitive data value or the address of an object instance
35 Version Control	The task of keeping a system consisting of many versions well organized

Figure 1: Will be referenced by questions in Multiple Choice section (next)

Multiple Choice

Read the full question and every possible answer. Choose the one best answer for each question and write the corresponding letter in the blank next to the number. The image below clarifies the meaning of open and closed arrows and diamonds. Note that "e.g." means "for example". {15 at 2 points each}

1. ____ **Operator == compares**

A. The memory address of two objects or primitives

B. the memory address of two objects, but the values of two primitives

C. The value of two objects or primitives

D. the value of two objects, but the memory address of two primitives

1. ____ **Operator** == **compares**

A. The memory address of two objects or primitives

B. the memory address of two objects, but the values of two primitives

C. The value of two objects or primitives

D. the value of two objects, but the memory address of two primitives

CORRECT: B

2. ____ **Class** `Book` **is an abstract class with a default constructor. Which of the following will create an instance of class** `Book`**?**

A. `Book book();`

B. `Book book = new Book();`

C. `Book book;`

D. An abstract class cannot be instanced

2. ____ **Class** `Book` **is an abstract class with a default constructor. Which of the following will create an instance of class** `Book`**?**

    A. `Book book();`

    B. `Book book = new Book();`

    C. `Book book;`

    D. An abstract class cannot be instanced

CORRECT: D

3. ____ **Which is TRUE about Java exceptions?**

A. An uncaught exception will cause the program to abort

B. The try clause may have one or more catch clauses

C. Java does NOT permit defining additional, custom exception types

D. An exception, once caught, cannot be rethrown

E. Any variable, even a primitive, may be thrown in Java (but throw exceptions please!)

F. An exception is a class in Java, and must be instanced when thrown

G. Only the main method can catch an exception

3. ____ **Which is TRUE about Java exceptions?**

A. An uncaught exception will cause the program to abort

B. The try clause may have one or more catch clauses

C. Java does NOT permit defining additional, custom exception types

D. An exception, once caught, cannot be rethrown

E. Any variable, even a primitive, may be thrown in Java (but throw exceptions please!)

F. An exception is a class in Java, and must be instanced when thrown

G. Only the main method can catch an exception

CORRECT: ABF

4. ____ **Which of the following ends the program and reports** `42` **to the operating system?**

A. `System.return(42);`
B. `return(42);`
C. `exit 42;`
D. `exit(42);`
E. `System.exit(42);`
F. `return 42;`

4. ____ **Which of the following ends the program and reports** `42` **to the operating system?**

A. `System.return(42);`

B. `return(42);`

C. `exit 42;`

D. `exit(42);`

E. `System.exit(42);`

F. `return 42;`

CORRECT: E

5. ____ **A successful regression test should print**

A. "1"

B. Nothing

C. "Pass"

D. "0"

5. ____ **A successful regression test should print**

A. "1"

B. Nothing

C. "Pass"

D. "0"

CORRECT: B

6. ____ **A final class**

A. cannot be edited

B. cannot be a subclass

C. cannot be instanced

D. cannot be a superclass

6. ____ **A final class**

A. cannot be edited

B. cannot be a subclass

C. cannot be instanced

D. cannot be a superclass

CORRECT: D

7. ____ **A final method**

A. cannot be overridden

B. cannot be edited

C. cannot modify fields

D. is called when an object is deleted

7. ____ **A final method**

A. cannot be overridden

B. cannot be edited

C. cannot modify fields

D. is called when an object is deleted

CORRECT: A

8. ____ **A final field**

A. cannot be accessed from subclasses

B. exists at the class level (one address for all objects)

C. cannot be edited

D. cannot be modified once constructed

8. ____ **A final field**

A. cannot be accessed from subclasses

B. exists at the class level (one address for all objects)

C. cannot be edited

D. cannot be modified once constructed

CORRECT: D

9. ____ **A Java subclass may inherit from**

A. One superclass and any number of interfaces

B. Either a superclass or an interface

C. Any number of superclasses and interfaces

D. Both one superclass and one interface

9. ____ **A Java subclass may inherit from**

A. One superclass and any number of interfaces

B. Either a superclass or an interface

C. Any number of superclasses and interfaces

D. Both one superclass and one interface

CORRECT: A

10. ____ **Documentation for Java packages is usually written using**

A. README.txt files in the same directory

B. GitHub Markdown

C. Javadoc

D. Microsoft Word

10. ____ **Documentation for Java packages is usually written using**

A. README.txt files in the same directory

B. GitHub Markdown

C. Javadoc

D. Microsoft Word

CORRECT: C

11. ____ **To control how an object is converted into a String, for example as a parameter to `System.println`, we would**

    A. Write a `format` function that returns the string representation

    B. Use Javadoc

    C. Specify an `sprintf` format string as a template

    D. Override the `toString()` method

11. ____ **To control how an object is converted into a String, for example as a parameter to `System.println`, we would**

    A. Write a `format` function that returns the string representation

    B. Use Javadoc

    C. Specify an `sprintf` format string as a template

    D. Override the `toString()` method

CORRECT: D

12. ____ **The difference between** `System.out` **and** `System.err` **is**

A. `System.out` doesn't check for errors, while `System.err` does

B. `System.out` streams out data, while `System.err` streams out error messages

C. `System.out` uses `println`, while `System.err` uses `printf`

D. They both do exactly the same thing

12. ____ **The difference between** `System.out` **and** `System.err` **is**

A. `System.out` doesn't check for errors, while `System.err` does

B. `System.out` streams out data, while `System.err` streams out error messages

C. `System.out` uses `println`, while `System.err` uses `printf`

D. They both do exactly the same thing

CORRECT: B

13. ____ **Which statement is TRUE about** `String` **in Java?**

A. The length of `String s` is determined using `strlen(s)`

B. `String` is a primitive type in Java

C. `StringBuilder` is MUCH faster than `String` when making a lot of changes

D. String is "immutable" and thus cannot be changed once constructed

E. Strings s1 and s2 may be concatenated into a single String using `s1 + s2`

F. A new `String` *must* be constructed using `String s = new String("flea");`

G. In Java, `String` is another name for `char*`

H. Chars may be accessed in `String s` using `for(char c : s.toCharArray())`

I. *Any* object `x` may be converted to a String using `x.toString()`

J. A specific `char` within a `String` may be accessed with subscripts like `s[3]`

K. Each `char` in a `String` is 8 bits, just like in a C `char*`

13. ____ **Which statement is TRUE about** `String` **in Java?**

A. The length of `String s` is determined using `strlen(s)`

B. `String` is a primitive type in Java

C. `StringBuilder` is MUCH faster than `String` when making a lot of changes

D. String is "immutable" and thus cannot be changed once constructed

E. Strings s1 and s2 may be concatenated into a single String using `s1 + s2`

F. A new `String` *must* be constructed using `String s = new String("flea");`

G. In Java, `String` is another name for `char*`

H. Chars may be accessed in `String s` using `for(char c : s.toCharArray())`

I. *Any* object `x` may be converted to a String using `x.toString()`

J. A specific `char` within a `String` may be accessed with subscripts like `s[3]`

K. Each `char` in a `String` is 8 bits, just like in a C `char*`

CORRECT: CDEHI

14. ____ **The class version of Java's array is the**

A. `ArrayClass`
B. `ListClass`
C. `Array`
D. `List`
E. `ArrayList`

14. ____ **The class version of Java's array is the**

A. `ArrayClass`

B. `ListClass`

C. `Array`

D. `List`

E. `ArrayList`

CORRECT: E

15. ____ **Refer to Figure 1 on page 2. The relationship between** `Clothing` **and** `Shirt` **is**

A. Association Class

B. Dependency

C. Association

D. Composition

E. Inheritance

F. Aggregation

15. ____ **Refer to Figure 1 on page 2. The relationship between** `Clothing` **and** `Shirt` **is**

A. Association Class

B. Dependency

C. Association

D. Composition

E. Inheritance

F. Aggregation

CORRECT: E

16. ____ **Refer to Figure 1 on page 2. The relationship between** `Clothing` **and** `ID` **is**

A. Composition

B. Aggregation

C. Association

D. Dependency

E. Inheritance

F. Association Class

16. ____ **Refer to Figure 1 on page 2. The relationship between** `Clothing` **and** `ID` **is**

A. Composition

B. Aggregation

C. Association

D. Dependency

E. Inheritance

F. Association Class

CORRECT: A

17. ____ **Refer to Figure 1 on page 2. The relationship between** `Shirt` **and** `Style` **is**

A. Aggregation

B. Association Class

C. Dependency

D. Composition

E. Inheritance

F. Association

17. ____ **Refer to Figure 1 on page 2. The relationship between** `Shirt` **and** `Style` **is**

A. Aggregation

B. Association Class

C. Dependency

D. Composition

E. Inheritance

F. Association

CORRECT: A

18. ____ **Refer to Figure 1 on page 2. The relationship between** `Size` **and** `Style` **is**

A. Inheritance

B. Aggregation

C. Composition

D. Association

E. Dependency

F. Association Class

18. ____ **Refer to Figure 1 on page 2. The relationship between** `Size` **and** `Style` **is**

A. Inheritance

B. Aggregation

C. Composition

D. Association

E. Dependency

F. Association Class

CORRECT: D

19. ____ **Refer to Figure 1 on page 2. The** `style` **field for class** `Shirt` **is**

A. Package-private

B. Public

C. Private

D. Protected

19. ____ **Refer to Figure 1 on page 2. The** `style` **field for class** `Shirt` **is**

A. Package-private

B. Public

C. Private

D. Protected

CORRECT: C

20. ____ **Refer to Figure 1 on page 2. The** `size` **field for class** `Style` **is**

A. Private

B. Package-private

C. Public

D. Protected

20. ____ **Refer to Figure 1 on page 2. The** `size` **field for class** `Style` **is**

A. Private

B. Package-private

C. Public

D. Protected

CORRECT: A

21. ____ **Refer to Figure 1 on page 2. The** `id` **field for class** `Clothing` **is**

A. Public

B. Package-private

C. Private

D. Protected

21. ____ **Refer to Figure 1 on page 2. The** `id` **field for class** `Clothing` **is**

A. Public

B. Package-private

C. Private

D. Protected

CORRECT: B

22. ____ **Refer to Figure 1 on page 2. The** `discard` **method for class** `Clothing` **is**

A. Protected

B. Public

C. Private

D. Package-private

22. ____ **Refer to Figure 1 on page 2. The** `discard` **method for class** `Clothing` **is**

A. Protected

B. Public

C. Private

D. Package-private

CORRECT: D

23. ____ **Refer to Figure 1 on page 2. The** `price` **field for class** `Clothing` **is**

A. Public

B. Protected

C. Private

D. Package-private

23. ____ **Refer to Figure 1 on page 2. The** `price` **field for class** `Clothing` **is**

A. Public

B. Protected

C. Private

D. Package-private

CORRECT: B

24. ____ **Refer to Figure 1 on page 2. The** `wash()` **method for class** `Clothing` **is**

A. Private

B. Protected

C. Public

D. Package-private

24. ____ **Refer to Figure 1 on page 2. The** `wash()` **method for class** `Clothing` **is**

A. Private

B. Protected

C. Public

D. Package-private

CORRECT: B

25. ____ **Refer to Figure 1 on page 2. The** `toString()` **method for class** `Shirt` **is**

A. Protected

B. Package-private

C. Private

D. Public

25. ____ **Refer to Figure 1 on page 2. The** `toString()` **method for class** `Shirt` **is**

A. Protected

B. Package-private

C. Private

D. Public

CORRECT: D

26. ____ **Refer to Figure 1 on page 2. For method** `toString()` **in class** `Shirt`**,** `<<override>>` **is a**

A. Constraint

B. Comment

C. Stereotype

D. Tag

26. ____ **Refer to Figure 1 on page 2. For method** `toString()` **in class** `Shirt`**,** `<<override>>` **is a**

A. Constraint

B. Comment

C. Stereotype

D. Tag

CORRECT: C

27. ____ **Refer to Figure 1 on page 2. The "{year=2025}" under class** `Style` **is a**

A. Constraint

B. Comment

C. Tag

D. Stereotype

27. ____ **Refer to Figure 1 on page 2. The "{year=2025}" under class** `Style` **is a**

A. Constraint

B. Comment

C. Tag

D. Stereotype

CORRECT: C

28. ____ **Refer to Figure 1 on page 2. The "{isMadeInUSA}" under class** `Shirt` **is a**

A. Stereotype

B. Tag

C. Constraint

D. Comment

28. ____ **Refer to Figure 1 on page 2. The "{isMadeInUSA}" under class** `Shirt` **is a**

A. Stereotype

B. Tag

C. Constraint

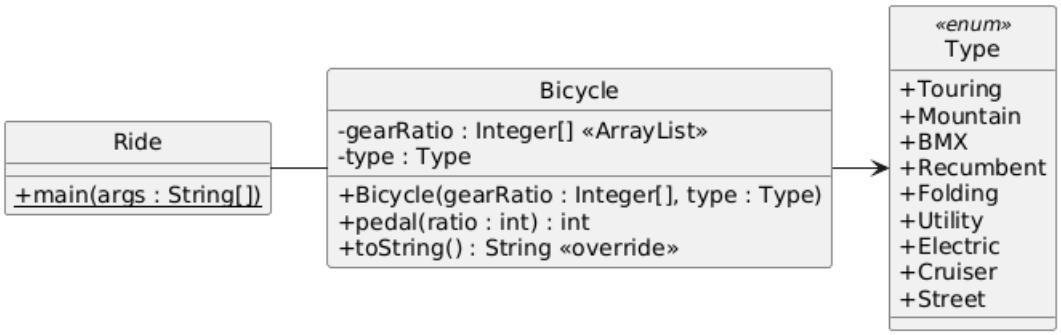
D. Comment

CORRECT: C

Free Response

Write the solutions to the Free Response questions in the provided space. Additional coding sheets are available on request. **Write your name and student ID on EVERY additional coding sheet you use that is not already stapled to this exam.**

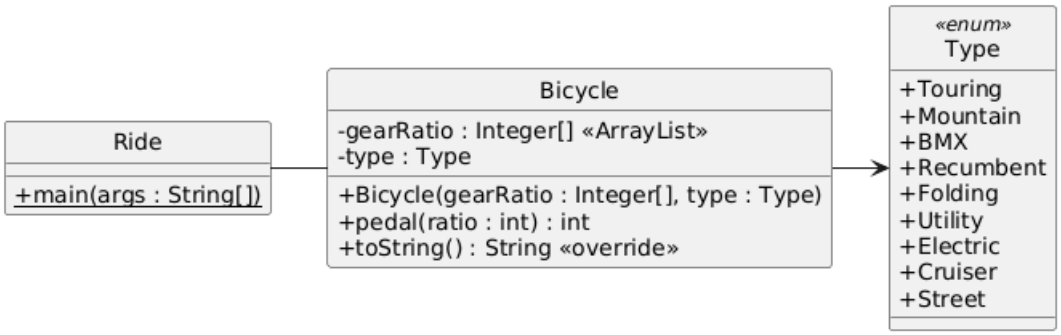
Each question stands alone, with instructions on what to code from the class diagram. If you have trouble answering a question, simply move to the next question and come back later. **Assume all needed imports - don't code them.**



NOTE: Students were shown only 3 of the Type enumerations.

1. {code an enum, 3 points} In file Type.java, code ONLY enum `Type` from the class diagram above so that it would be visible in all packages in the application.

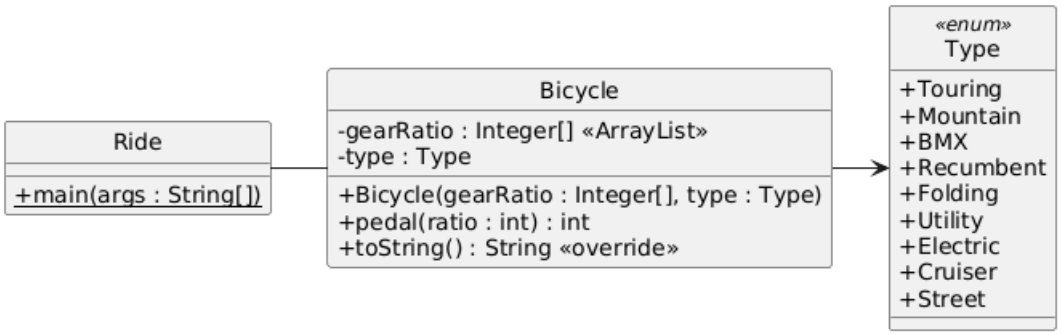
```
public enum Type {Touring,   Mountain, BMX,
                  Recumbent, Folding,  Utility,
                  Electric,  Cruiser,  Street} // Subset of 3 of these
```



2. {code a class} For this question, consider the `Bicycle` class in the class diagram above.

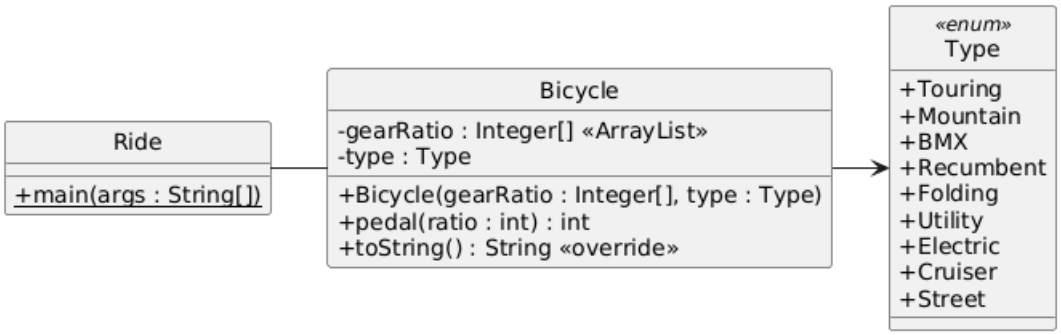
a. {5 points} In file Bicycle.java, write the class (such that it is visible across all packages) and field declaration but omit the methods (they are covered below).

```
public class Bicycle {
   private ArrayList<Double> gearRatio;
   private Type type;
```



b. {6 points} In file Bicycle.java, write the `Bicycle` class constructor. Note that `gearRatio` is an `ArrayList`. If `gearRatio` is `null` OR empty, throw an `IllegalArgumentException` with the message "Bad gearbox". Otherwise, assign each field to its corresponding parameter.

```
public Bicycle(Type type, ArrayList<Double> gearRatio) {
     if(gearRatio == null || gearRatio.isEmpty())
         throw new IllegalArgumentException("Bad gearbox");
     this.gearRatio = gearRatio;
     this.type = type;
 }
```

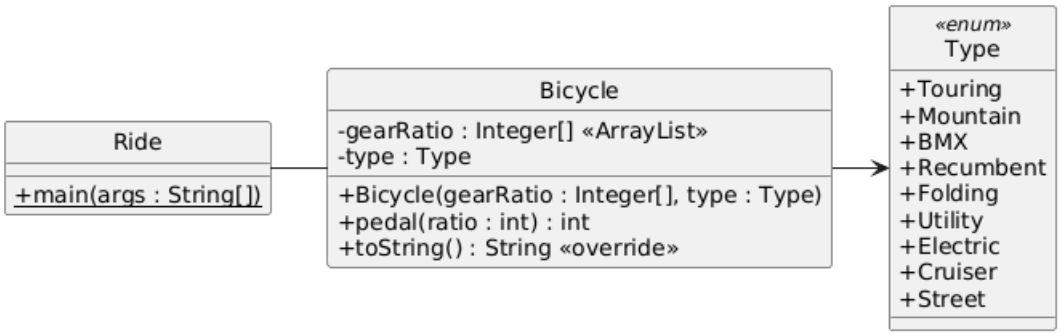


c. {5 points} In file Bicycle.java, write the `pedal` method. If the parameter `gear` (an `int`) is a valid index for ArrayList `gearRatio`, return the ratio value (a `Double`) at index `gear` in `gearRatio`, otherwise return [an error code].

```
public double pedal(int gear) {
    if(gear<0 || gear >= gearRatio.size()) return -1; // code varies
    return gearRatio.get(gear);
}
```

Conversely,

```
public double pedal(int gear) {
    if(gear>=0 && gear < gearRatio.size()) return gearRatio.get(gear);
    return -1; // code varies
}
```



NOTE: Students were shown only 3 of the Type enumerations. One of the 3 Type enumerations was selected at random for the example below.

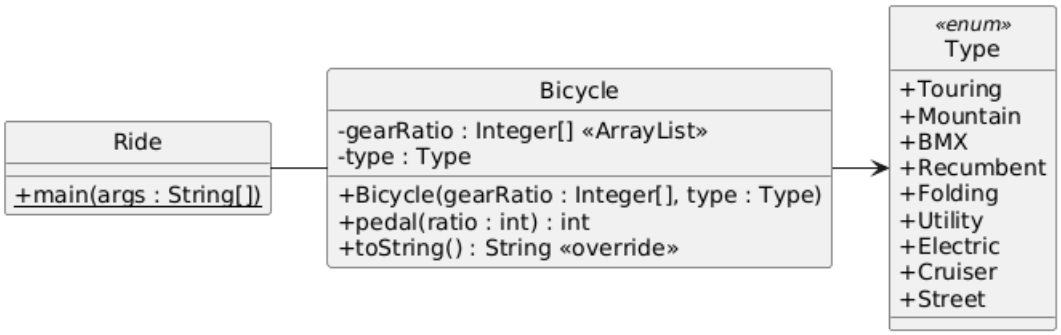
d. {7 points} In file Bicycle.java, override the `toString()` method such that a compiler error will be generated if the superclass has no matching method.

Using a `StringBuilder` object, return the `type` of bicycle, number of gears, and the `gearRatio` values separated by commas (for ANY number of gears).

For example, a Folding bike with gear ratios of 1.1, 2.5, 4.2, and 9.9 would return

```
Folding 4-speed with ratios 1.1, 2.5, 4.2, 9.9
```

```java
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("" + type);
    sb.append(" " + gearRatio.size() + "-speed with ratios ");
    String separator = "";
    for(double ratio : gearRatio) {
        sb.append(separator + ratio);
        separator = ", ";
    }
    return sb.toString();
}
```



NOTE: Students were shown only 3 of the Type enumerations. One of the three Type enumerations, three random ratios, and a random return code were selected at random for method main to instance.

3. {code a main method, 8 points} For this question, code the `Ride` class in the class diagram above. Do NOT write imports; assume you have what you need.

In the main method, instance a `Bicycle` of type `Cruiser` with gear ratios 1.5, 3.6, and 5.7.

Print your `Bicycle` instance.

Then call method `pedal` on it, selecting the middle gear (gear 1), and print to the standard out stream "Pedaling at ratio " and the result of method `pedal`.

If an `IllegalArgumentException` occurs at any point during the above, print its message to the standard error stream and exit the program with -1.

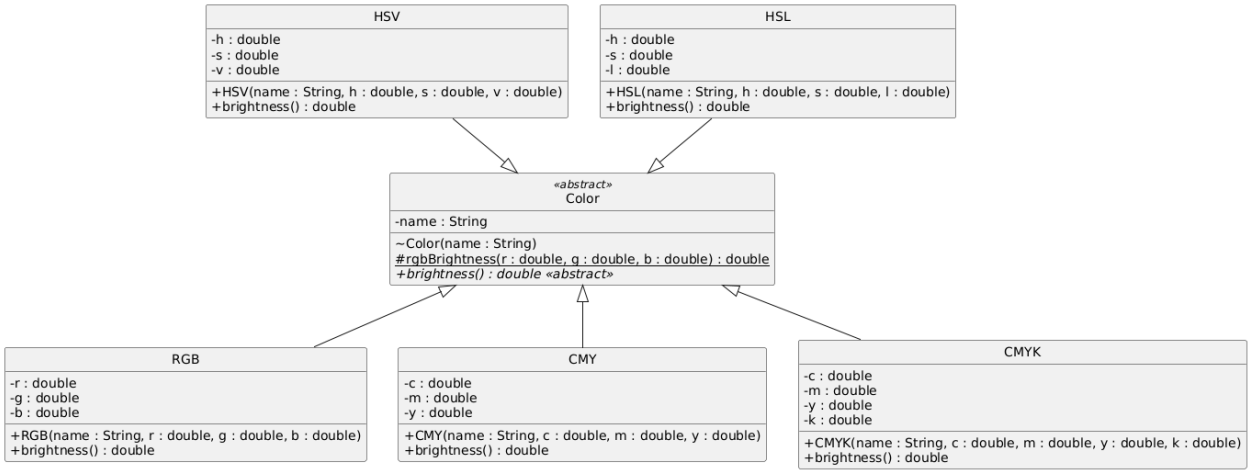
If correctly written, your program should have the following output:

```
Cruiser 3-speed with ratios 1.5, 3.6, 5.7
Pedaling at ratio 3.6
```

Here's one example for Type.Cruiser and ratios 1.5, 3.6, and 5.7 (your Types and ratios may vary):

```
public class Ride {
   public static void main(String[] args) {
       ArrayList<Double> gearBox = new ArrayList<>();
       gearBox.add(1.5); gearBox.add(3.6); gearBox.add(5.7); // ratios vary
       try {
           Bicycle b = new Bicycle(Type.Cruiser, gearBox); // Type varies
           System.out.println(b);
           System.out.println("Pedaling at ratio " + b.pedal(1));
       } catch(IllegalArgumentException e) {
           System.err.println(e);
           System.exit(-1); // Code varies
       }
   }
}
```

4. {inheritance} Consider the class diagram below.

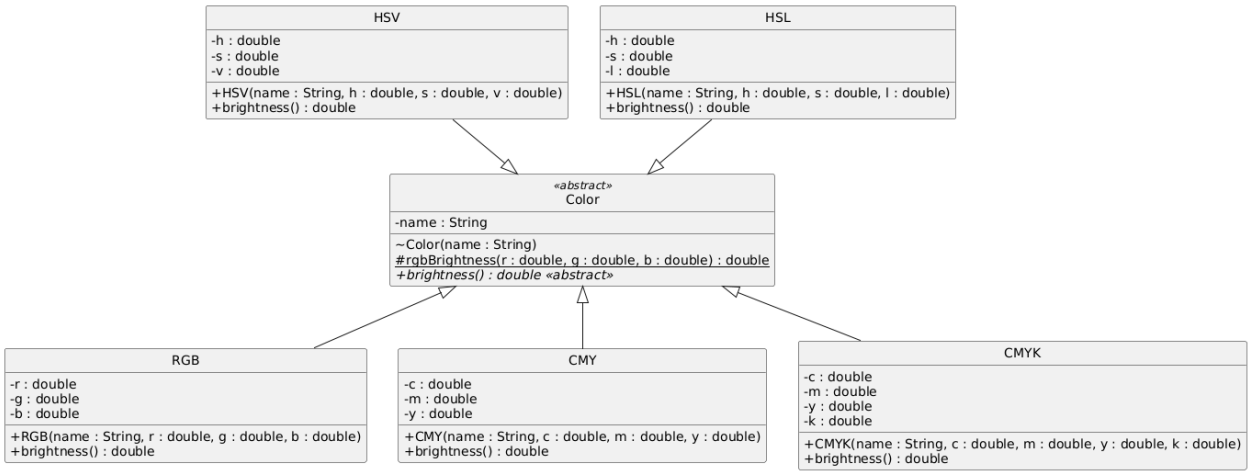


NOTE: One of the subclasses was randomly selected for each student to display with subclass `CMYK`.

a. {7 points} In file Color.java, write all of abstract class `Color`. The package-private constructor simply assigns its parameter to the field. Static method `rgbBrightness` returns the double value `0.2126*r + 0.7152*g + 0.0722*b`. Also code `brightness`.

```
public abstract class Color {
   Color(String name) {
        this.name = name;
   }
   protected static double rgbBrightness(double r, double g, double b) {
        return 0.2126*r + 0.7152*g + 0.0722*b;
   }
   public abstract double brightness();

   private String name;
}
```

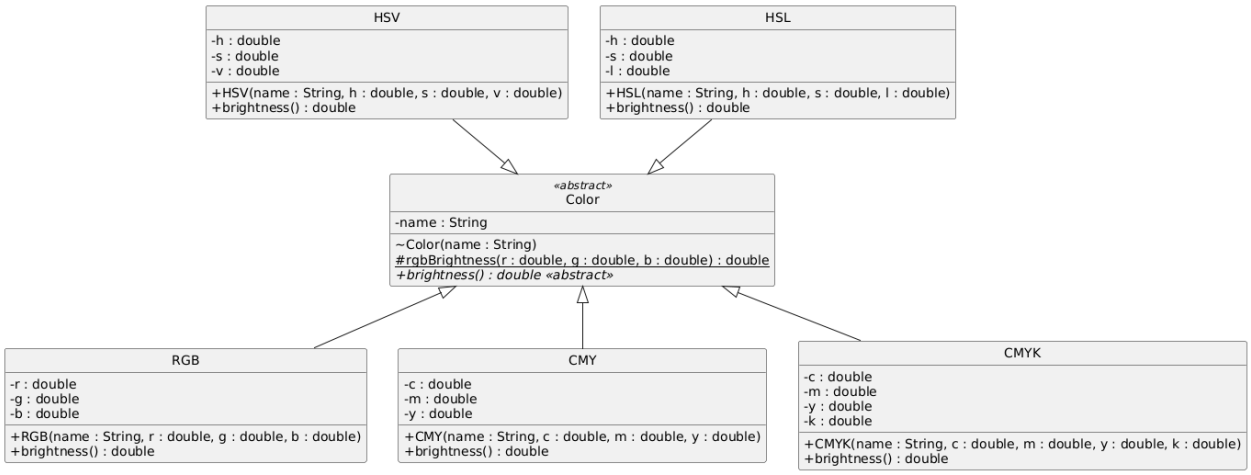


NOTE: The randomly selected subclass (not CMYK) was requested. A suggested solution for RGB is given below.

b. {9 points} In file RBG.java, write all of the subclass RGB. The constructor constructs all inherited and local fields from the parameters per Java requirements.

The `brightness` method (for which the compiler MUST verify is being overridden) must return the result of `Color`'s static method `rgbBrightness(r, g, b)`.

```
public class RGB extends Color {
   public RGB(String name, double r, double g, double b) {
       super(name);
       this.r = r;
       this.g = g;
       this.b = b;
   }
   @Override
   public double brightness() {
       // super.rgbBrightness OK, Color.rgbBrightness wrong but no deduction
       return rgbBrightness(r, g, b);
   }
   private double r;
   private double g;
   private double b;
}
```

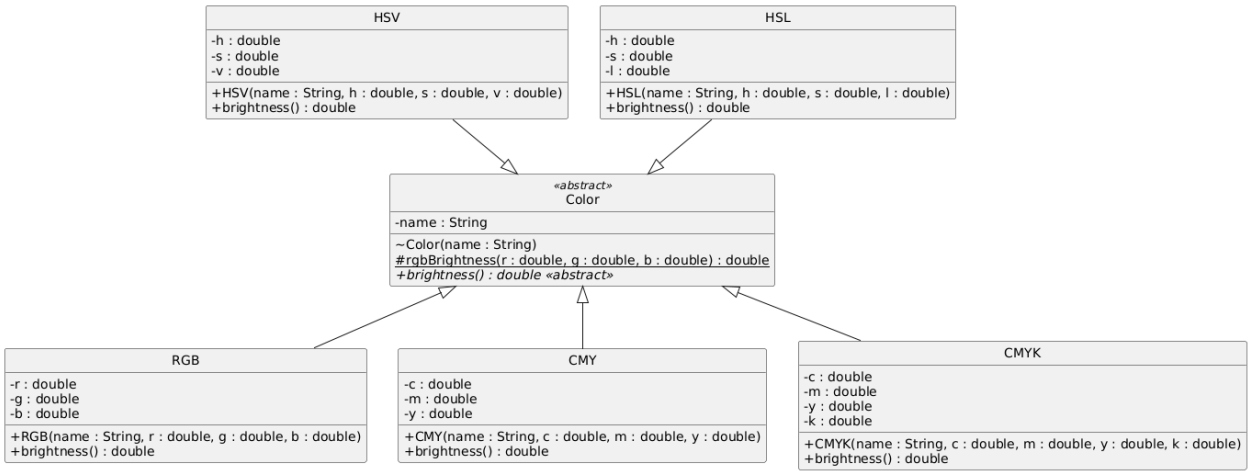


NOTE: The randomly selected subclass (not `CMYK`) was requested. A suggested solution for `HSV` is given below.

b. {9 points} In file HSV.java, write all of the subclass `HSV`. The constructor constructs all inherited and local fields from the parameters per Java requirements.

The `brightness` method (for which the compiler MUST verify is being overridden) must return the result of `Color`'s static method `rgbBrightness(v/0.0709, v/0.2384, v/0.0241)`.

```
public class HSV extends Color {
   public HSV(String name, double h, double s, double v) {
       super(name);
       this.h = h;
       this.s = s;
       this.v = v;
   }
   @Override
   public double brightness() {
       // super.rgbBrightness OK, Color.rgbBrightness wrong but no deduction
       return rgbBrightness(v/0.0709, v/0.2384, v/0.0241);
   }
   private double h;
   private double s;
   private double v;
}
```

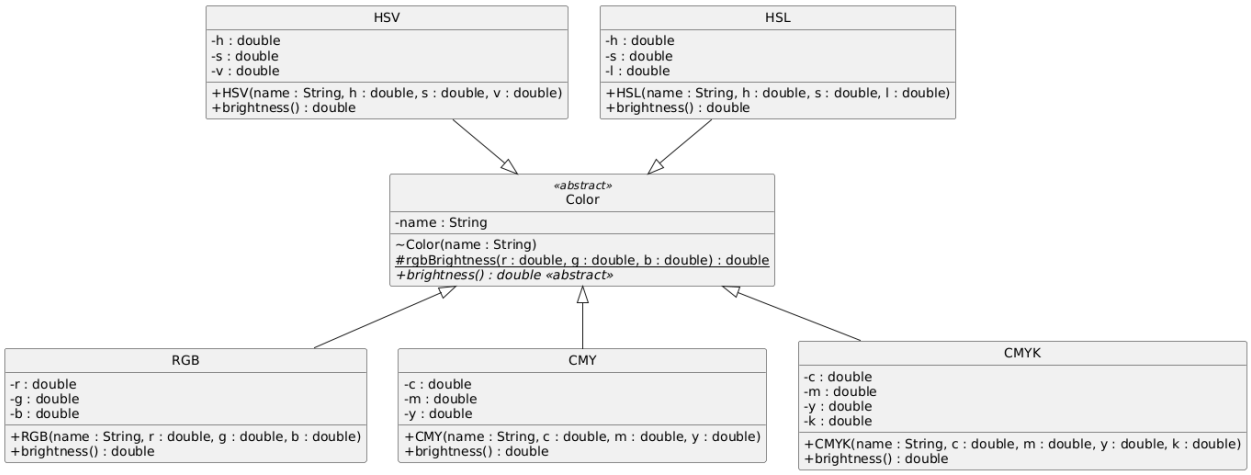


NOTE: The randomly selected subclass (not CMYK) was requested. A suggested solution for HSL is given below.

b. {9 points} In file HSL.java, write all of the subclass HSL. The constructor constructs all inherited and local fields from the parameters per Java requirements.

The `brightness` method (for which the compiler MUST verify is being overridden) must return the result of `Color`'s static method `rgbBrightness(l/0.0709, l/0.2384, l/0.0241)` (those are els, not ones!).

```
public class HSL extends Color {
   public HSL(String name, double h, double s, double l) {
       super(name);
       this.h = h;
       this.s = s;
       this.l = l;
   }
   @Override
   public double brightness() {
       // super.rgbBrightness OK, Color.rgbBrightness wrong but no deduction
       return super.rgbBrightness(l/0.0709, l/0.2384, l/0.0241);
   }
   private double h;
   private double s;
   private double l;
}
```



NOTE: The randomly selected subclass (not `CMYK`) was requested. A suggested solution for `CMY` is given below.

b. {9 points} In file CMY.java, write all of the subclass `CMY`. The constructor constructs all inherited and local fields from the parameters per Java requirements.

The `brightness` method (for which the compiler MUST verify is being overridden) must return the result of `Color`'s static method `rgbBrightness(1-c, 1-m, 1-y)`.

```
public class CMY extends Color {
   public CMY(String name, double c, double m, double y) {
       super(name);
       this.c = c;
       this.m = m;
       this.y = y;
   }
   @Override
   public double brightness() {
       // super.rgbBrightness OK, Color.rgbBrightness wrong but no deduction
       return rgbBrightness(1-c, 1-m, 1-y);
   }
   private double c;
   private double m;
   private double y;
}
```

Bonus

BONUS 1: {+4 points} `String s` has size of at least 3 chars. Demonstrate up to 4 *different* Java algorithms for determining if the first 3 characters of `s` are "CSE".

NOTE: By "demonstrate" we intended for you to write concise *code* (as in "Talk is cheap. Show me the *code*."). However, very clear and unambiguous explanations for an algorithm may have been awarded partial or full credit at the grader's discretion.

The last "algorithm" resulted in enough laughter that we gave full credit for it anyway. :)

We accepted these same algorithms when applied to a newly-constructed `StringBuilder` object as long as the code is valid. `StringBuilder` includes `indexOf`, `substring`, and `charAt` methods, but no `startsWith`, `split`, or `matches` method.

```
String[] strings = new String[]{"CSE1325", "NOT CSE1325"};
for(String s : strings) {
    if(s.startsWith("CSE"))           System.out.println(s + " Algorithm 1");
    if(s.indexOf("CSE") == 0)         System.out.println(s + " Algorithm 2");
    if(s.substring(0,3).equals("CSE")) System.out.println(s + " Algorithm 3");
    if(s.charAt(0) == 'C' &&
       s.charAt(1) == 'S' &&
       s.charAt(2) == 'E')            System.out.println(s + " Algorithm 4");
    if(s.split("CSE")[0].isEmpty())   System.out.println(s + " Algorithm 5");
    if(s.matches("^CSE.*"))           System.out.println(s + " Algorithm 6");
    System.out.println(s + " <-- does this start with 'CSE'?   Algorithm 7");
}
```

BONUS 2: {+3 points} In no more than TWO concise sentences, explain the difference between a permissive and a share-alike software license and give a major example license of each type.

A permissive license like MIT allows open source code to be included in proprietary apps.

A share-alike license like Gnu GPL requires all code added to the open source code to also use a compatible share-alike license.