Small Assignment #5
Due: Wednesday, 3/19/2025 by 11:59 PM

Submission. Submit this as a single PDF on Gradescope. Assign pages to questions.

Question 1.
In your own words, explain what it means that Java's type system is primarily *static*, relies mostly on *type annotations*, and is considered *strong*.

Question 2.
Define *method overloading*, and explain how Java's compiler determines (based on the source code) which method is being called.

Question 3.
What is the general rule for what kind of parameter coercion is allowed in Java? Give an example of it.

Question 4.
Consider a simple function called `addThree` for adding three numbers together. Note that I am not specifying a specific return type, but it should be a number. Let's say we want all of the following calls to compile and run in Java:

```
addThree(1, 2, 3);
addThree(1.1, 2.1, 3.1);
addThree(1, 2.2, 3.3);
addThree(1.1, 2, 3.3);
addThree(1, 2.2, 3.3);
```

(a) Explain how you could implement this in Java using only *method overloading.*
(b) Explain how you could implement this in Java using only *parameter coercion*.
(c) Explain how you could implement this in Java using a combination of *method overloading* and *parameter coercion.*

Question 5.
Consider the Java code below. First, predict the results of each print statement. You can check your answers, but you should try to figure it out yourself first because you may be asked similar questions on a quiz/exam. Second, explain what kinds of polymorphism are shown here. Be specific in your explanation – that is, don't just list the names of the polymorphism. Explain where they are in the code. Be very clear in your answer by referring to the line numbers.

```
1     public class Book {
2         private String title;
3         private String author;
4
5         public Book(String title, String author) {
```

```
6           this.title = title;
7           this.author = author;
8         }
9
10        public String getTitle() { return this.title; }
11        public String getAuthor() { return this.author; }
12
13        @Override
14        public String toString() {
15          return this.title + " by " + this.author;
16        }
17    }
18
19    public class SA5 {
20        public static void main(String[] args) {
21          int a = 1;
22          int b = 2;
23          double c = 3.3;
24          double d = 4.45;
25          String s = "apple";
26          String t = "banana";
27          Book b1 = new Book("It", "Stephen King");
28          Book b2 = new Book("Ulysses", "James Joyce");
29
30          System.out.println(a + c + s);
31          System.out.println((double) a + b / 2);
32          System.out.println((double) ((a + b)/2));
33          System.out.println(s + a + d + t);
34          System.out.println(s + (a + d) + t);
35          System.out.println(b1);
36          System.out.println(b2.getTitle());
37        }
38    }
```

Question 6.

Consider the Book class given above in Question 5. Each of the following is a proposed addition to the Book class for comparing the equality of two books based on their titles and authors. For each proposed addition, answer the following questions: (a) Will this code compile? Explain why or why not. (b) If the code compiles, is this an example of *overriding*, *overloading*, or neither? Explain your answer.

(1)

```
public boolean equals(Object obj) {
      if(obj == null) return false;
```

```
      if(obj == this) return true;
      if(obj.getClass() != this.getClass()) return false;
      return obj.title.equals(this.title) && obj.author.equals(this.author);
}
```

(2)

```
public boolean equals(Object obj) {
      if(obj == null) return false;
      if(obj == this) return true;
      if(obj.getClass() != this.getClass()) return false;
      return ((Book)obj).title.equals(this.title) &&
          ((Book)obj).author.equals(this.author);
}
```

(3)

```
@Override
public boolean equals(Object obj) {
      if(obj == null) return false;
      if(obj == this) return true;
      if(obj.getClass() != this.getClass()) return false;
      return ((Book)obj).title.equals(this.title) &&
          ((Book)obj).author.equals(this.author);
}
```

(4)

```
@Override
public boolean equals(Book other) {
      if(other == null) return false;
      if(other == this) return true;
      if(other.getClass() != this.getClass()) return false;
      return other.title.equals(this.title) &&
             other.author.equals(this.author);
}
```

(5)

```
public boolean equals(Book other) {
      if(other == null) return false;
      if(other == this) return true;
      if(other.getClass() != this.getClass()) return false;
      return other.title.equals(this.title) &&
             other.author.equals(this.author);
}
```

Question 7.
In your own words, explain what `@Override` does.

Question 8.
In your own words, explain why you should override the `hashCode` method if you override the `equals` method.