Q1 Inhertiance
21 Points

For each of the following, tell if the code will compile. If so, what will be be printed? If not, why will it not compile?

Give a one sentence reasoning for EITHER outcome. (No reasoning gets max 1 point.) Note: Syntax errors, if any, are inadvertent and should be ignored. Also, code outside of an explicit class is considered to be in the main method of some arbitrary class in the same package. (Fields that
are not private are accessible to all classes in same package.)

Q1.1
3 Points

```
public class Y {
   public int y1;
}

public class Z extends Y {
   public int z1;
}

Z z = new Y();
System.out.println(z.y1);
```

The code will not compile
- Y y = new Z(); this creates an object of Z. but assigned to a variable of it's base class Y.
so, we can access a from Y, but we can't access z1 from variable of Y type.

Q1.2
3 Points

```
public class P {
   public int a;
}
```

```
public class Q extends P {
    public int b;
}
P p = new Q();
System.out.println(p.b);
```

The code will not compile
- P p = new Q(); this creates an object of Q. but assigned to a variable of it's base class P.
so, we can access a from P, but we can't access b from variable of P type.

Q1.3
4 Points

```
public class A {
    public static int v=5;
}
public class B extends A {
    public int v;
}

System.out.println(B.v);
```

The code will not compile
-non-static variable v cannot be referenced from a static context

Q1.4
4 Points

```
public class X {
    public int x=2;
    public int getX() {
        return x;
    }
    public String toString() {
        return x + "";
    }
}
public class Y extends X {
    public int y=3;
    public String toString() {
        return (getX() + y) + "";
    }
}
X x = new Y();
System.out.println(x);
```

The code will compile, and print 5.
- The code will compile and print 5 because the toString method in the Y class will be called when the x variable is printed. The toString method in the Y class returns the sum of the x and y variables, which are 2 and 3 respectively. Therefore, the output will be 5.

Q1.5
4 Points

```
public class A {
    public int x;
    public A(int x) {
        this.x = x;
    }
}
public class B extends A {
     public static void main(String[] args) {
         System.out.println("B");
     }
}
```

- The code will not compile because the B class does not have a constructor.
- A class needs a constructor in order to be instantiated. The B class does not have a constructor, so it cannot be instantiated. Therefore, the code will not compile.

Q1.6
3 Points

```
public class M {
   public static int mn=2;
}
public class N extends M {
   public int mn;
}
M m = new N();
System.out.println(m.mn);
```

- The code will not compile because the mn field in the N class is not static.
- In order for the mn field in the N class to be accessible from the M class, it must be declared static. However, the mn field in the N class is not static, so the code will not compile.

Q2 Interfaces
16 Points

For each of the following, tell if the code will compile. If so, no explanation is needed. If not, explain why.

Syntax errors, if any, are inadvertent and should be ignored. Also, code outside of an explicit class is considered to be in the main method of some arbitrary class in the same package.

Q2.1
3 Points

```
public interface I<T> {
   void imethod(T t);
}
public class SomeI implements I<Integer> {
   public void imethod(SomeI sm) { }
}
```

It won't compile.

- The method imethod in SomeI must accept an Integer parameter since the interface I<T> was parameterized to Integer.

imethod(T t) becomes imethod(Integer t) because SomeI implements I<Integer>

Q2.2
3 Points

```
public interface H {
   void hm();
}
public class HI implements H {
   public void hm() { }
}
H hobj = new H();
```

It won't compile.

H is an interface not a class so you cannot instantiate it.

H hobj = new H();
-> this should be
H hobj = new HI();
because HI is a class so you can instantiate it.

Q2.3
4 Points

```
public interface A<E> {
   void astuff(E e);
}
public interface B<E> {
   void bstuff(E e);
}
public class AB implements A<AB>, B<AB> {
    ... // assume astuff/bstuff implemented
}
A<AB> intf = new AB();
intf.bstuff(new AB());
```

It won't compile.

A<AB> intf = new AB();
-> intf is declared as an A type parametered with AB type.
So calling intf.bstuff, it won't work since intf being A is supposed to only have method astuff.
Either you declare it as B(AB) intf, or as AB intf, to access bstuff.

Q2.4
3 Points

```
public class X implements Comparable<X> {
   public int compareTo(X x) {
      return 0;
   }
}
public class Z extends X implements Comparable<Z> {
   public int compareTo(Z z) {
      return 0;
   }
}
```

It won't compile.
- The interface Comparable cannot be implemented more than once with different arguments. At first, the comparable interface of type X already has been implemented. Implementing again with Type Z will raise an error.

Q2.5
3 Points

```
public class Student {
   ...
   public Student(String name) { ... }
   public int compareTo(Student s) { ... }
}

public class Library {
   public static <T extends Comparable<T>>
```

```
    void libm(T item) { ... }
}

Library.libm(new Student("Jane Doe"));
```

It won't compile.
-The method libm() takes a parameter of type T not of Type Student and the parameter passed in calling the method is of type Student.

Q3 Lambda Expressions
19 Points

You are given the following class definition (assume all methods are correctly implemented):

```
public class Song {
    ...
    public Song(String name, String artist, String genre)
     { ... }
    public Song(String name, String artist) { ... }
    public String getName() { ... }
    public String getArtist() { ... }
    public String getGenre() { ... }
    public int copiesSold() { ... }
}
```

For each of the following, write NAMED and TYPED lambda expressions. In other words, LHS (left hand side) is a type and a variable name, and RHS (right hand side) is the lambda expression. For the type, use appropriate functional interfaces from the java.util.function and java.util packages. (Only the lambda expression with no type+name will get max half credit):

Q3.1
4 Points

Get the genre of a song (do NOT use a method reference)

Function<Song,String> songGenreGetter = (song)->song.getGenre();

Q3.2
4 Points

A method reference to create a Song instance instance with name and artist

```
Predicate<Song> isPop = song ->
song.getGenre().equals("Pop");
Function<Song, String> songName = Song::getName;
Function<Song, String> artistName = song -> song.getArtist();
BiFunction<String, String, Song> songCreator = (name, artist) -
> new Song(name, artist);
Supplier<Song> songSupplier = () -> new Song("title", "artist",
"genre");
Consumer<Song> songPrinter = System.out::println;
```

Q3.3
6 Points

A predicate for songs that are not of the genre "Pop" or "Rock", and have sold 10,000 or more copies. You may write named and typed supporting predicates, if needed.

```
Predicate<Song> equalToPop = (i) -> i.getGenre() == "Pop";
Predicate<Song> equalToRock = (i) -> i.getGenre() == "Rock";
Predicate<Song> copiesMoreThanTenThousand = (i) ->
i.copiesSold() >= 10000;

boolean res=
((equalToPop.or(equalToRock)).negate()).and(copiesMoreThanTen
Thousand).test(obj);
```

Q3.4
5 Points

An expression whose LHS variable can be passed as argument to the sort method of a List<Song> of songs, for sorting in ascending order of copies sold.

```
Comparator<Song> sortByCopiesSold = (s1, s2) ->
s1.copiesSold() - s2.copiesSold();
```

Q4 Class Polymorphism
7 Points

Recall the Point and ColoredPoint classes discussed in lecture. Suppose there is an array Point[] pts that is populated with a mix of Point and ColoredPoint objects. Write a loop to print the color of every ColoredPoint instance in this array, using the ColoredPoint class's getColor() method. Is your code polymorphic? Argue for why or why not.

```
for(Point p : points) {
if(p instanceof ColoredPoint) {
ColoredPoint c = (ColoredPoint) p;

System.out.println(c.getColor());
   }
}
```

If parent and chlild having the same method, then it would be polymorphis.

Q5 Interface Polymorphism
6 Points

There are two kinds of interface polymorphism. Explain each with the use of Java snippets to show polymorphic behavior.
(Your code doesn't have to implement any algorithmic logic, just the essential set up needed to support your explanation.)

1. Run time polymorphism
- The Java virtual machine, and not the Java compiler, is responsible for resolving this sort of polymorphism, which is also resolved to as Dynamic Method Dispatch. Because of this, the form of polymorphism in question is referred to as run-time polymorphism.
class Machine {

```
    public void start() {
        System.out.println("Machine have moving parts.");
    }
}

class Jet extends Machine {
    public void start() {
        System.out.println("Jets use rotating engines.");
    }
}

public class TestJet {

    public static void main(String args[]) {
        Machine a = new Machine(); // Machine reference and
object
        Machine b = new Jet(); // Machine reference but Jet
object

        a.start();// runs the method in Machine class
        b.start();//Runs the method in Jet class
    }
}
```

2. Compile-time polymorphism
- Compile-time polymorphism can be illustrated with the use of the method known as method overloading. The term "overloading" refers to a function that shares the same name but has a distinct signature.

```
class A
{
        void m1()
        {
                System.out.println("Inside A's m1 method");
        }
}
class B extends A
{
        void m1()
        {
                System.out.println("Inside B's m1 method");
        }
}
class C extends A
{
        void m1()
        {
                System.out.println("Inside C's m1 method");
        }
}
class CallingMethod
{
        public static void main(String args[])
        {
                A a=new A(); //creating object of class A
                B b=new B(); //creating object of class B
                C c=new C(); //creating object of class C
                A ref; //creating reference variable of class A
                ref=a; //assigning object of class A to reference
variable of class A
                ref.m1(); //calling m1 method of class A
                ref=b; //assigning object of class B to reference
variable of class A
                ref.m1(); //calling m1 method of class B
                ref=c; //assigning object of class C to reference
variable of class A
                ref.m1(); //calling m1 method of class C
        }
}
```

Q6 equals Method
6 Points

Suppose there is class called X. Assume the following objects are created:

```
Object o = new X();
X x = new X();
```

In class X, a public boolean equals(X x) method is implemented. Which version of equals: equals(Object) or equals(X), is called in each of the following? Give a 1-2 sentence explanation for your answer.

Q6.1
3 Points

o.equals(o);

equals(Object) is called when o.equals(o); is used.
Because o is of type Object.

Q6.2
3 Points

x.equals(x);

equals(X) is called when x.equals(x); is used.
Because x is of type X.

Midterm Exam

● Graded

Student
KANGHWI LEE

Total Points
63 / 75 pts

Question 1
Inhertiance 18 / 21 pts

1.1	(no title)	3 / 3 pts
1.2	(no title)	3 / 3 pts

1.3	(no title)	4 / 4 pts
1.4	(no title)	4 / 4 pts
1.5	(no title)	4 / 4 pts
1.6	(no title)	0 / 3 pts

Question 2
Interfaces — 16 / 16 pts

2.1	(no title)	3 / 3 pts
2.2	(no title)	3 / 3 pts
2.3	(no title)	4 / 4 pts
2.4	(no title)	3 / 3 pts
2.5	(no title)	3 / 3 pts

Question 3
Lambda Expressions — 19 / 19 pts

3.1	(no title)	4 / 4 pts
3.2	(no title)	4 / 4 pts
3.3	(no title)	6 / 6 pts
3.4	(no title)	5 / 5 pts

Question 4
Class Polymorphism — 4 / 7 pts

Question 5
Interface Polymorphism — 0 / 6 pts

Question 6
equals Method — 6 / 6 pts

6.1	(no title)	3 / 3 pts
6.2	(no title)	3 / 3 pts