Assignment 6

1. what is method overloading in Java ~~Q~~ & explain with an example ?

Ans → (i) method overloading allows you to define multiple methods with name in a class, as long as their parameter lists differ.

(ii) This enables you to create methods ~~that with the~~ ~~[illegible]~~ ~~[illegible] as a class~~, that performs similar operations but accept different types or number of arguments.

(iii) Example:

```
class Addition {
        public int add (int a, int b) {
            return a + b;
        }

        public double add(double a, double b) {
            return a+b;
        }
}
```

(iv) In this example, Calculator class has 2 add methods, where one takes two int arguments & returns their sum & the other one takes two double arguments & returns their sum.

2. what are the rules for method overloading resolution in Java? How does Java determine which overloaded method to call ?

Ans → Java uses following rules to determine which overloaded method to call:

(i) No. of arguments: No. of arguments passed in call must match exactly with one of the method def^n^s.

(ii) Argument types: The types arguments must be compatible by either exact matches or promotable through widening conversions.

(iii) Return type is not considered.

When multiple methods match based on first two rules, Java selects the most specific method based on argument types is considered more specific than one with less specific types.

3. What does the static keyword mean in Java? Explain the difference b/w static & non-static methods.

Ans → (i) The static keyword in Java is used to create a class members (methods, variables) that belong to the class itself, not to the class itself, not to individuals objects of the class. ~~static keyword~~

(ii) static members exist not only once in memory & are shared by all instance of class.

(iii) Difference b/w Static & Non-static methods:

- Non-static methods:
  - It requires an object reference to be called.
  - They can access both static & non-static members of the class.

- Static methods:
  - It can be called directly using class name w/o the reference to be called.
  - They can access only static members of class & not non-static members directly.

4. Can static methods be overloaded & overridden in Java? How are static variables shared across multiple instances of a class.

Ans → (i) Static methods can be overloaded like non-static methods based on parameter lists.

(ii) Static methods cannot be overridden in subclasses because overriding requires inheritance & polymorphism which doesn't apply to static methods.

(iii) Static variables are shared across all instances of a class because they exist in a single memory location.

(iv) Any instances can modify the value of a static variable, and the change will be reflected for all be reflected for all other instances as well.

5. What is the role of the static keyword in the context of memory management?

Ans → ~~The~~ (i) The static keyword doesn't directly affect memory mgmt. in Java.

(ii) However, because static variables are loaded into memory when the class is loaded, they are available throughout the entire program execⁿ.

(iii) Non-static (instance) variables are created when an object is instantianized & are garbaged collected when the object is no longer referenced.

6. What is the significance of final keyword in Java?

Ans → (i) The final keyword in Java is used to restrict modifications.

(ii) It can be applied to:

#4

- Variables: Declared a variable that cannot be reassigned a new value after initializaⁿ.
- Methods: Prevents the method from being overridden in subclasses.
- Classes: Prevents the class from being subclassed.

(ii) Effects of final:
- final variables: Ensures consistent values throughout the program.
- final methods: Guarantees that the behavior of the method remains the same in subclasses.
- final classes: Prevents inheritance, making the class the final form.

7. Can a final ~~keyword~~ method be overridden in a subclass? How does the final keyword affect variables, methods, & classes in Java?

Ans → (i) final methods cannot be overridden in subclasses. This ensures that method's behavior remains unchanged in the hierarchy.

8. What does this keyword represent in Java? How is this keyword used in constructors & methods?

Ans → (i) this keyword refers to the current object instance.

(ii) Its used in following contexts:

- Constructors: to access & initialize instance variables ~~[illegible]~~ within the constructor.
- Methods: To differentiate instance variables with the same name. It allows you to access instance variables within a method.

(iii) Example:

```
class Person {
    ~~pass~~ private String name;
        public ~~P~~ Person (String name) {
            this.name = name; // use this.name to
                    distinguish from the constructor parameter
        }
        public void introduce() {
            System.out.println("Hello, my name is" +
                this.name);
        }
}
```

9. What are narrowing & widening conversions in Java?

Ans ⇒ (i) Narrowing Conversions: Converting a value from wider data type to a narrower data type.
This can potentially lead to data loss if the wider data type's value cannot be accurately represented in the narrower type.

(ii) widening Conversions: Converting a value from a wider data type.
This is generally safe as no data loss occurs.

10. Provide examples of narrowing & widening conversions b/w primitive data types.

Ans → (i) ~~[illegible]~~ Narrowing:
- double to float, long, int, short or byte
- long to int short or byte
- float to int, short or byte
- int to short or byte.

(ii) widening:
- byte to short, int, long, float or double
- long to int, short, or byte
- float to int, short or byte
- int to short or byte

11. How does Java handle potential loss of precision during narrowing in Java?

Ans → (i) Java attempts to represents the value from the wider Type as closely as possible in the narrower type.
(ii) However, during narrowing conversions, data loss can occur if the wider type's value exceeds the range of narrower type.
- For narrower type Java truncates the decimal portion when converting to int type.
Ex: double d = 3.14;
int i = (int) d; // i will be 3

- In extreme cases, the value might not be converted to the min^n & max^m value of the narrower type.

12. Explain the concept of automatic widening conversions in Java?

Ans → (i) Java automatically performs widening conversions when necessary during expressions & assignments.

(ii) This happens when you mix operands of different types in an operation, and the result needs to be a wider type to accomodate both values.

(iii) Ex:

~~byte b=10;~~
~~short~~

```
byte b = 10;
short s = 20;
int sum = b + s;  // widening conversion of
                     byte to int before addition.
```

13. What are the implications of narrowing ~~&~~ & widening conversions on type compatibility and data loss?

Ans → (i) Type Compalibity: Widening conversions generally improve compatibility as they allow mixing data types in expressions.

(ii) Data Loss: Narrowing conversions can lead to data loss if not handled carefully.
We should be aware of the potential loss of precision when conversions from to a narrower type.