1. How to create an Object in Java?
   Answer: We use the ‘new’ keyword to create an object.
   Syantax : ClassName objectName = new ClassName();

2. What is the use of ‘new’ keyword in Java?
   Answer: The ‘new’ keyword in Java is used to dynamically allocate memory for an object at runtime. It creates an instance of a class, invokes it’s constructor, and returns a reference to the object.

3. What are the different types of variables in Java?
   Answer: There are three types of variables in Java based on behavior and position of the declaration: They are:
   a. Instance Variable
   b. Local Variable
   c. Static Variable

4. What is the difference between Instance Variable and Local Variable?
   Answer:
   a. Scope: Instance variables have a class-level scope and are accessible from any method in the class, while local variables have a method-level scope and are only accessible with the method in which they are declared.
   b. Initialization: Instance variables are automatically initialized to their default values(e.g. 0 for numeric types, false for boolean, and null for reference types), while local variables must be explicitly initialized before they can be used.
   c. Lifetime: Instance variables persist as long as the object they belong to exists while local variables are created and destroyed every time the method in which they are declared is called.
   d. Accessibility: Instance variables can be accessed from outside the class using getter and setter methods, while local variables cannot be accessed outside the method in which they are declared.

```
public class MyClass {
        int x; // instance variable

        public void myMethod() {
                int y; // local variable
         }
}
```

5. In which area of memory is allocated for Instance Variable and Local Variable?
   Answer: In Java, Instance Variables are allocated in heap memory, while Local Variables are stored in the stack memory area.

6. What is method overloading?

Answer: Method Overloading in Java is a feature that allows a class to have two or more methods with the same name, as long as they have different parameters. This is useful when you want to provide multiple ways of performing the same action with different inputs.

When a class has multiple methods with the same name, the Java compiler determines which method to call at compile-time based on the number and types of the argument passed in the method call. This is known as compile-time polymorphism or static polymorphism.

Example:

```
class ClassName {
        // Overloaded methods
        public void methodName(int x) { ... }
        public void methodName(double x) { ... }
        public void methodName(int x, double y) { ... }
  ...
}
```