Day 10 PW Skills Assignment

1. **Ans.**	**How to create an object in java?** ClassName objectName = new ClassName();
2. **Ans.**	**What is the use of a new keyword in java?** In Java, the new keyword is used to create an instance of a class or to dynamically allocate memory for an object. 1. Memory allocation: The new keyword allocates memory on the heap to hold the object's data and state. The amount of memory allocated depends on the size and structure of the class. 2. Object initialization: Once the memory is allocated, the new keyword calls the constructor of the class to initialize the object. The constructor sets up the initial state of the object and performs any necessary initialization tasks. 3. Object reference: The new keyword returns a reference to the newly created object. This reference can be assigned to a variable, allowing you to access and manipulate the object using that variable.
3. **Ans.**	**What are the different types of variables in Java?** In Java, variables are categorized into several types based on their scope, lifetime, and purpose. Here are the different types of variables in Java: 1. Local Variables: These variables are declared and used within a block or method. They have a limited scope and exist only within the block or method where they are declared. Local variables must be initialized before they are used. 2. Instance Variables (Non-Static Fields): Instance variables are declared within a class but outside any method, constructor, or block. They are associated with an instance or object of the class and have separate copies for each instance. Instance variables are initialized with default values if not explicitly initialized. 3. Class Variables (Static Fields): Class variables are declared with the static keyword and are associated with the class itself, rather than instances of the class. They have only one copy shared among all instances of the class. Class variables are initialized with default values if not explicitly initialized. 4. Parameters: Parameters are variables declared in method or constructor signatures. They receive values when the method or constructor is called and provide a way to pass values into methods or constructors.

4. **What is the difference between Instance variable and Local variable?**

Ans.

	Instance Variables	**Local Variables**
Scope	Class-level scope	Block-level scope
Lifetime	Exist as long as the object exists	Exist only within the block of code
Initialization	May or may not be explicitly initialized	Must be explicitly initialized before use
Memory Allocation	Allocated memory when object is created	Allocated memory on the stack when block is executed
Accessibility	Accessible throughout the class	Accessible only within the block
Default Values	Assigned default values if not explicitly initialized	No default values, must be initialized before use

5. **In which area memory is allocated for instance variable and local variable?**

Ans. In Java, memory allocation for instance variables and local variables is done in different areas.

1. Instance Variables: Memory for instance variables is allocated on the heap. The heap is a region of memory where objects are dynamically allocated. Each instance variable is associated with a specific object instance, and memory is allocated for the instance variables when the object is created using the new keyword. Each object has its own copy of instance variables.

	2. Local Variables: Memory for local variables is allocated on the stack. The stack is a region of memory used for storing local variables, method parameters, and other temporary data during method or block execution. Local variables are created when the block in which they are declared is executed, and their memory is automatically released when the block is exited or their execution scope ends.
6. **Ans.**	**What is method overloading?** Method overloading in Java refers to the ability to define multiple methods in a class with the same name but with different parameters. In method overloading: 1. Method name remains the same and parameter differs. 2. Return type is not considered: Overloaded methods can have the same or different return types. 3. Access modifiers, exceptions, and modifiers can be the same or different: Overloaded methods can have the same or different access modifiers (e.g., public, private, protected), exceptions thrown, or other modifiers (e.g., static, final). Java determines which overloaded method to invoke at compile-time based on the arguments provided during method invocation. The compiler matches the method call with the appropriate method based on the parameter types, number, and order.