Programs and Explanation

1. Factors of a Number

Steps to Find Factors:

1. Start from 1: Begin by checking from the number 1(i).
2. Divide and Check: Check if the number is divisible by the current number (i). If it is, then it is a factor.
3. Continue: Continue this process up to the number itself.

Example:

To find the factors of 12:

- 1 (12 is divisible by 1)
- 2 (12 is divisible by 2)
- 3 (12 is divisible by 3)
- 4 (12 is divisible by 4)
- 6 (12 is divisible by 6)
- 12 (12 is divisible by 12)

Therefore, the factors of 12 are: 1, 2, 3, 4, 6, 12

Code:

```
import java.util.Scanner;


public class Factors {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Ask the user to enter a number
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
```

```
        System.out.println("Factors of " + number + " are:");

        // Loop to find and print factors

        for (int i = 1; i <= number; i++) {

            if (number % i == 0) {

                System.out.print(i + " ");

            }

        }

    }

}
```

2. Prime Number

Definition:

A prime number is a number greater than 1 that has no divisors other than 1 and itself. In other words, a prime number is only divisible by 1 and itself.

Steps to Check if a Number is Prime:

1. Input the Number: Start by taking input from the user.
2. Initialize Counter: Initialize a counter variable to zero. This will be used to count the number of divisors.
3. Loop Through Numbers: Use a for-loop to iterate from 1 to the entered number.
4. Check for Divisors: In each iteration, check if the current number (i) divides the entered number without leaving a remainder.
5. Increment Counter: If the current number is a divisor, increment the counter by 1.
6. Determine Primality: After the loop, check the value of the counter.
    - If the counter is exactly 2 (meaning the number has only two divisors: 1 and itself), the number is prime.
    - If the counter is not 2, the number is not prime.
7. Display Result: Print whether the number is prime or not based on the counter's value.

Example:

- **2**: Divisible by 1 and 2 (Prime)
- **3**: Divisible by 1 and 3 (Prime)
- **4**: Divisible by 1, 2, and 4 (Not Prime)
- **5**: Divisible by 1 and 5 (Prime)
- **6**: Divisible by 1, 2, 3, and 6 (Not Prime)

Code:

```
import java.util.Scanner;


public class PrimeNumberSimple {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Ask the user to enter a number

        System.out.print("Enter a number: ");

        int number = scanner.nextInt();


        int count = 0;  // Counter for number of divisors


        for (int i = 1; i <= number; i++) {

            if (number % i == 0) {

                count++;

            }

        }


        // If the count is exactly 2, the number is prime

        if (count == 2) {

            System.out.println(number + " is a prime number.");
```

```
        } else {
            System.out.println(number + " is not a prime number.");
        }
    }
}
```

3. Composite Number

Definition:

A **composite number** is a number greater than 1 that is not a prime number. In other words, a composite number has more than two divisors.

Steps to Check if a Number is Composite:

1. Input the Number: Start by taking input from the user.
2. Initialize Counter: Initialize a counter variable to zero. This will be used to count the number of divisors.
3. Loop Through Numbers: Use a for-loop to iterate from 1 to the entered number.
4. Check for Divisors: In each iteration, check if the current number (i) divides the entered number without leaving a remainder.
5. Increment Counter: If the current number is a divisor, increment the counter by 1.
6. Determine Compositeness: After the loop, check the value of the counter.
    - If the counter is more than 2 (meaning the number has more than two divisors), the number is composite.
    - If the counter is 2 or less, the number is not composite.
7. Display Result: Print whether the number is composite or not based on the counter's value.

Example:

- **4**: Divisible by 1, 2, and 4 (Composite)
- **6**: Divisible by 1, 2, 3, and 6 (Composite)
- **9**: Divisible by 1, 3, and 9 (Composite)
- **10**: Divisible by 1, 2, 5, and 10 (Composite)

Code:

```
import java.util.Scanner;

public class CompositeNumber {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Ask the user to enter a number
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        int count = 0;  // Counter for number of divisors

        for (int i = 1; i <= number; i++) {
            if (number % i == 0) {
                count++;
            }
        }

        // If the count is more than 2, the number is composite
        if (count > 2) {
            System.out.println(number + " is a composite number.");
        } else {
            System.out.println(number + " is not a composite number.");
        }
    }
```

}

4. Perfect Number

Definition:

A **perfect number** is a positive integer that is equal to the sum of its proper divisors, excluding itself. Proper divisors are all positive divisors of the number other than the number itself.

Steps to Check if a Number is Perfect:

1. Input the Number: Start by taking input from the user.
2. Initialize Sum: Initialize a variable to store the sum of proper divisors.
3. Loop Through Possible Divisors: Use a for-loop to iterate from 1 to one less than the entered number.
4. Check for Divisors: In each iteration, check if the current number (i) divides the entered number without leaving a remainder.
5. Sum Divisors: If the current number is a divisor, add it to the sum.
6. Determine Perfection: After the loop, check if the sum of the divisors is equal to the entered number.
7. Display Result: Print whether the number is perfect or not based on the sum's value.

Example:

- 6: The proper divisors of 6 are 1, 2, and 3. The sum is 1+2+3=6. Thus, 6 is a perfect number.
- 28: The proper divisors of 28 are 1, 2, 4, 7, and 14. The sum is 1+2+4+7+14=28. Thus, 28 is a perfect number.
- 496: The proper divisors of 496 are 1, 2, 4, 8, 16, 31, 62, 124, and 248. The sum is 1+2+4+8+16+31+62+124+248=496. Thus, 496 is a perfect number.

Code:

```
import java.util.Scanner;


public class PerfectNumber {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Ask the user to enter a number
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        int sum = 0;  // Sum of proper divisors

        for (int i = 1; i < number; i++) { // Loop runs till less than the number
            if (number % i == 0) {
                sum += i;
            }
        }

        // If the sum of proper divisors is equal to the number, it is a perfect number
        if (sum == number) {
            System.out.println(number + " is a perfect number.");
        } else {
            System.out.println(number + " is not a perfect number.");
        }
    }
}
```

5. Abundant Number

Definition:

An **abundant number** is a number for which the sum of its proper divisors (excluding itself) is greater than the number itself.

Steps to Check if a Number is Abundant:

1. **Input the Number**: Start by taking input from the user.
2. **Initialize Sum**: Initialize a variable to store the sum of proper divisors.
3. **Loop Through Possible Divisors**: Use a for-loop to iterate from 1 to one less than the entered number.
4. **Check for Divisors**: In each iteration, check if the current number (i) divides the entered number without leaving a remainder.
5. **Sum Divisors**: If the current number is a divisor, add it to the sum.
6. **Determine Abundance**: After the loop, check if the sum of the divisors is greater than the entered number.
7. **Display Result**: Print whether the number is abundant or not based on the sum's value.

Example:

- 12: The proper divisors of 12 are 1, 2, 3, 4, and 6. The sum is 1+2+3+4+6=16. Since 16 is greater than 12, 12 is an abundant number.
- 18: The proper divisors of 18 are 1, 2, 3, 6, and 9. The sum is 1+2+3+6+9=21. Since 21 is greater than 18, 18 is an abundant number.
- 20: The proper divisors of 20 are 1, 2, 4, 5, and 10. The sum is 1+2+4+5+10=22. Since 22 is greater than 20, 20 is an abundant number.

Code:

```
import java.util.Scanner;


public class AbundantNumber {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Ask the user to enter a number

        System.out.print("Enter a number: ");
```

```
        int number = scanner.nextInt();


        int sum = 0;  // Sum of proper divisors


        for (int i = 1; i < number; i++) { // Loop runs till less than the number
            if (number % i == 0) {
                sum += i;
            }
        }


        // If the sum of proper divisors is greater than the number, it is an abundant
    number
        if (sum > number) {
            System.out.println(number + " is an abundant number.");
        } else {
            System.out.println(number + " is not an abundant number.");
        }
    }
}
```

6. Deficient Number

Definition:

A **deficient number** is a number for which the sum of its proper divisors (excluding itself) is less than the number itself.

Steps to Check if a Number is Deficient:

1. **Input the Number**: Start by taking input from the user.
2. **Initialize Sum**: Initialize a variable to store the sum of proper divisors.
3. **Loop Through Possible Divisors**: Use a for-loop to iterate from 1 to one less than the entered number.
4. **Check for Divisors**: In each iteration, check if the current number (i) divides the entered number without leaving a remainder.
5. **Sum Divisors**: If the current number is a divisor, add it to the sum.
6. **Determine Deficiency**: After the loop, check if the sum of the divisors is less than the entered number.
7. **Display Result**: Print whether the number is deficient or not based on the sum's value.

Example:

- **8**: The proper divisors of 8 are 1, 2, and 4. The sum is 1+2+4=7. Since 7 is less than 8, 8 is a deficient number.
- **14**: The proper divisors of 14 are 1, 2, and 7. The sum is 1+2+7=10. Since 10 is less than 14, 14 is a deficient number.
- **21**: The proper divisors of 21 are 1, 3, and 7. The sum is 1+3+7=11. Since 11 is less than 21, 21 is a deficient number.

Code:

```
import java.util.Scanner;


public class DeficientNumber {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Ask the user to enter a number

        System.out.print("Enter a number: ");

        int number = scanner.nextInt();


        int sum = 0;  // Sum of proper divisors
```

```
        for (int i = 1; i < number; i++) { // Loop runs till less than the number
            if (number % i == 0) {
                sum += i;
            }
        }

        // If the sum of proper divisors is less than the number, it is a deficient number
        if (sum < number) {
            System.out.println(number + " is a deficient number.");
        } else {
            System.out.println(number + " is not a deficient number.");
        }
    }
}
```

7. Pronic Number

Definition:

A **pronic number** (also known as a rectangular number, oblong number, or heteromecic number) is a number that is the product of two consecutive integers, that is, it is of the form n(n + 1).

Steps to Check if a Number is Pronic:

1. Input the Number: Start by taking input from the user.

2. Initialize Variable: Initialize a variable `fact` to zero. This will be used to check if the number is pronic.
3. Loop Through Numbers: Use a for-loop to iterate through numbers starting from 1 up to the entered number.
4. Check for Divisors: For each number `i`, check if the entered number is divisible by `i`.
5. Nested Check for Pronic Condition: Inside the first condition, check if $i \times (i+1)$ equals the entered number. If true, set `fact` to `i` and break the loop.
6. Determine Pronic: If `fact` is not zero after the loop, the number is pronic.
7. Display Result: Print whether the number is pronic or not based on the value of `fact`.

Example:

- **6**: 2×3=6, so 6 is a pronic number.
- **12**: 3×4=12, so 12 is a pronic number.
- **20**: 4×5=20, so 20 is a pronic number.
- **30**: 5×6=30, so 30 is a pronic number.

Code:

```
import java.util.Scanner;


public class PronicNumber {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Ask the user to enter a number

        System.out.print("Enter a number: ");

        int number = scanner.nextInt();


        int fact = 0;  // Variable to check pronic condition


        // Loop through numbers from 1 to the entered number

        for (int i = 1; i <= number; i++) {
```

```
            if (number % i == 0) {
                if (i * (i + 1) == number) {
                    fact = i;
                    break;
                }
            }
        }

        // Print the result
        if (fact != 0) {
            System.out.println(number + " is a pronic number.");
        } else {
            System.out.println(number + " is not a pronic number.");
        }
    }
}
```