**Task1**

Statically Typed Language: In a statically typed language, the data type of a variable is known at compile-time. This means that the type of value a variable can hold is determined before the program is executed. Java is a statically typed language.

Dynamically Typed Language: In a dynamically typed language, the data type of a variable is determined at runtime. This allows for more flexibility but can lead to runtime errors. Python is an example of a dynamically typed language.

Strongly Typed Language: In a strongly typed language, type compatibility is strictly enforced, and type conversions are explicit. This ensures that operations are performed on compatible data types only. Java is a strongly typed language.

Loosely Typed Language: In a loosely typed language, type compatibility is not as strict, and implicit type conversions are often performed. This can lead to unexpected behavior if not managed carefully. JavaScript is an example of a loosely typed language.

**Task 2**

Case Sensitive: Case-sensitive means that the language distinguishes between uppercase and lowercase characters. For example, "Hello" and "hello" are considered different in a case-sensitive language.

Case Insensitive: Case-insensitive means that the language treats uppercase and lowercase characters as the same. For example, "Hello" and "hello" are considered equal in a case-insensitive language.

Case Sensitive-Insensitive: This term doesn't exist in programming terminology. It seems to be a combination of the previous two concepts.

Java is a case-sensitive language, meaning "variable" and "Variable" would be treated as different variables.

**Task 3.**

Identity Conversion: Identity conversion occurs when a value is assigned to a variable of the same type without requiring any explicit casting. For example:

```
int num = 42;
char letter = 'A';
```

**Task 4.**

Primitive Widening Conversion: Primitive widening conversion occurs when a value of a smaller data type is automatically converted to a value of a larger data type. This is done to avoid loss of data. For example:

```
int intValue = 10;
double doubleValue = intValue;
```

**Task 5.**

Run-time Constant: A run-time constant is a value that is known only during the program's execution. It is evaluated at runtime, and its value can be determined based on user input or other runtime factors.

Compile-time Constant: A compile-time constant is a value that can be determined at compile time and remains constant throughout the program's execution. It's typically used for optimization purposes.

```
final int compileTimeConstant = 5;
final int runTimeConstant = someValueFromUserInput;
```

**Task6.**

Implicit Narrowing Primitive Conversions: These occur when a value of a larger data type is assigned to a variable of a smaller data type. Java allows this only if the value can be represented without loss of data.

Explicit Narrowing Conversions (Casting): These involve explicitly specifying that you're converting from one type to another using casting. It's used to inform the compiler that you're aware of potential data loss.

Implicit narrowing conversions are allowed when the value can fit into the smaller data type. Explicit narrowing conversions (casting) require a type cast operator and are subject to potential data loss.

**Task7.**

A long data type in Java is 64 bits, while a float data type is 32 bits. When assigning a long to a float, the value might lose precision, but Java allows this conversion as long as the range of the long value fits within the range of the float value. This is because floating-point numbers can represent a larger range of values, albeit with reduced precision for very large or very small values.

**Task8.**

`int` and `double` were set as the default data types for integer literals and floating-point literals in Java due to their common usage and to ensure compatibility with other programming languages. `int` is used for integers because it's a common choice for whole numbers, and `double` is used for floating-point literals because it provides a good balance between precision and range for most real-world applications.

**Task9.**

Implicit narrowing primitive conversion only takes place among `byte`, `char`, `int`, and `short` because these data types are commonly used for numeric operations, and the conversions between them are often meaningful. They cover a broad range of numeric values and are therefore more likely to be used together.

**Task10.**
Widening Primitive Conversion: This occurs when a value of a smaller data type is automatically converted to a larger data type. This doesn't result in loss of information. For example, converting an `int` to a `long`.

Narrowing Primitive Conversion: This occurs when a value of a larger data type is assigned to a variable of a smaller data type, potentially resulting in data loss. For example, converting a `double` to an `int`.

The conversion from `short` to `char` is not classified as both widening and narrowing because it doesn't involve a change in size. Both `short` and `char` have the same size (16 bits), so it's more accurately described as a direct conversion rather than a widening or narrowing conversion.