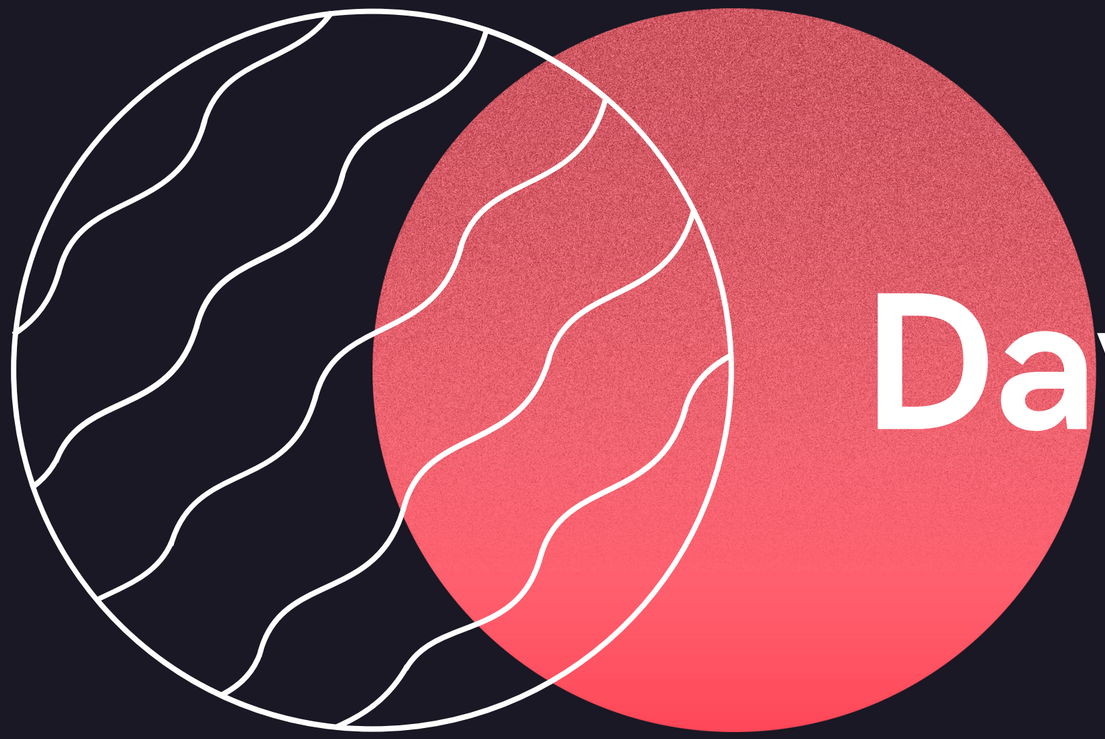


Day 1

Content

- Access modifiers
- static and instance
- type casting
- return types
- boolean expression

Access Modifiers:-

Access Modifiers essentially help define scope in a particular class.
Scope is the area in a program in which an object can be accessed.

There are 3 types of access modifiers:-

PUBLIC **PRIVATE** **PROTECTED**

OVERVIEW

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

private

With a private access modifier, the function will not be accessible outside a particular class.

```
class AccessModifiers
{
    private int priv(int a,int b)
    {
        return a+b;
    }
}
```

Given here is a class AccessModifiers, with a private function priv. The task is to return a sum

When this is accessed by another class, it throws a warning that says the method is NOT VISIBLE!

```
class Main
{
    public static vo
    {
        AccessModifi
        System.out.println(p.priv(1,2));
    }
}

class AccessModifiers
{
```

The method priv(int, int) from the type AccessModifiers is not visible Java(67108965

int AccessModifiers.priv(int a, int b)

View Problem No quick fixes available

ERROR

```
priv(int,int) has private access in AccessModifiers
```

public

With a public access modifier, the object will be accessible ANYWHERE!

```
class Main
{
    public static void main(String[] args)
    {
        AccessModifiers p = new AccessModifiers();
        System.out.println(p.pub(1,2));
    }
}

class AccessModifiers
{
   public int pub(int a,int b)
   {
        return a+b;
   }
}
```

Here no error or warning is thrown,
because we know for a fact that the
public function 'pub' is accessible anywhere! This applies to outside the given package too!

protected

With a protected access modifier, the object will be kinda be accessible?

```
class Main
{
    public static void main(String[] args)
    {
        AccessModifiers ac = new AccessModifiers();
        System.out.println(ac.prot(1,2));
    }
}

class AccessModifiers
{
   protected int prot(int a,int b)
   {
        return a+b;
   }
}
```

This program will work just fine. The function can be called in any class within the SAME PACKAGE!
This will not work for a different package

Exercise 1:-

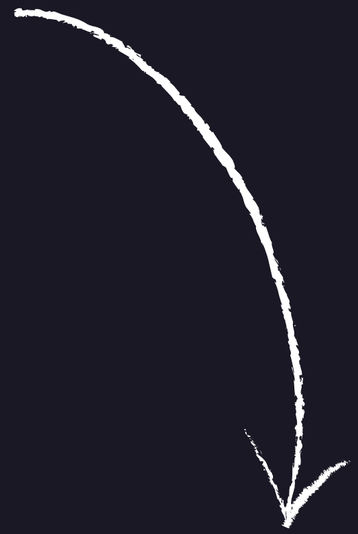
Create a package 'protpack' and create a protected object/function.

Import this package from a different program, and try to call/use this function/object

ETA: 3-5 minutes

TYPE CASTING

NARROWING



WIDENING

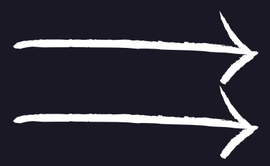
NARROWING TYPE CASTING

converting a larger type to
a smaller size type

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  `double` -> `float` -> `long` -> `int` -> `char` -> `short` -> `byte`

```
public static void narrowingCasting()
{
    double pi = 3.14;
    int a = (int) pi;

    System.out.println(pi);
    System.out.println(a);
}
```



```
3.14
3
```

WIDENING TYPE CASTING

converting a smaller type to
a larger size type

Widening Casting (automatically) - converting a smaller type to a larger type size

`byte` -> `short` -> `char` -> `int` -> `long` -> `float` -> `double`

```java
public static void wideningCast()
{
    byte first = 2;
    int second = first;

    System.out.println(second);
}
```

BOOLEAN EXPRESSION

Operator	Description
==	Returns true if the expression on the left evaluates to the same value as the expression on the right.
!=	Returns true if the expression on the left does not evaluate to the same value as the expression on the right.
<	Returns true if the expression on the left evaluates to a value that is less than the value of the expression on the right.
<=	Returns true if the expression on the left evaluates to a value that is less than or equal to the expression on the right.
>	Returns true if the expression on the left evaluates to a value that is greater than the value of the expression on the right.
>=	Returns true if the expression on the left evaluates to a value that is greater than or equal to the expression on the right.