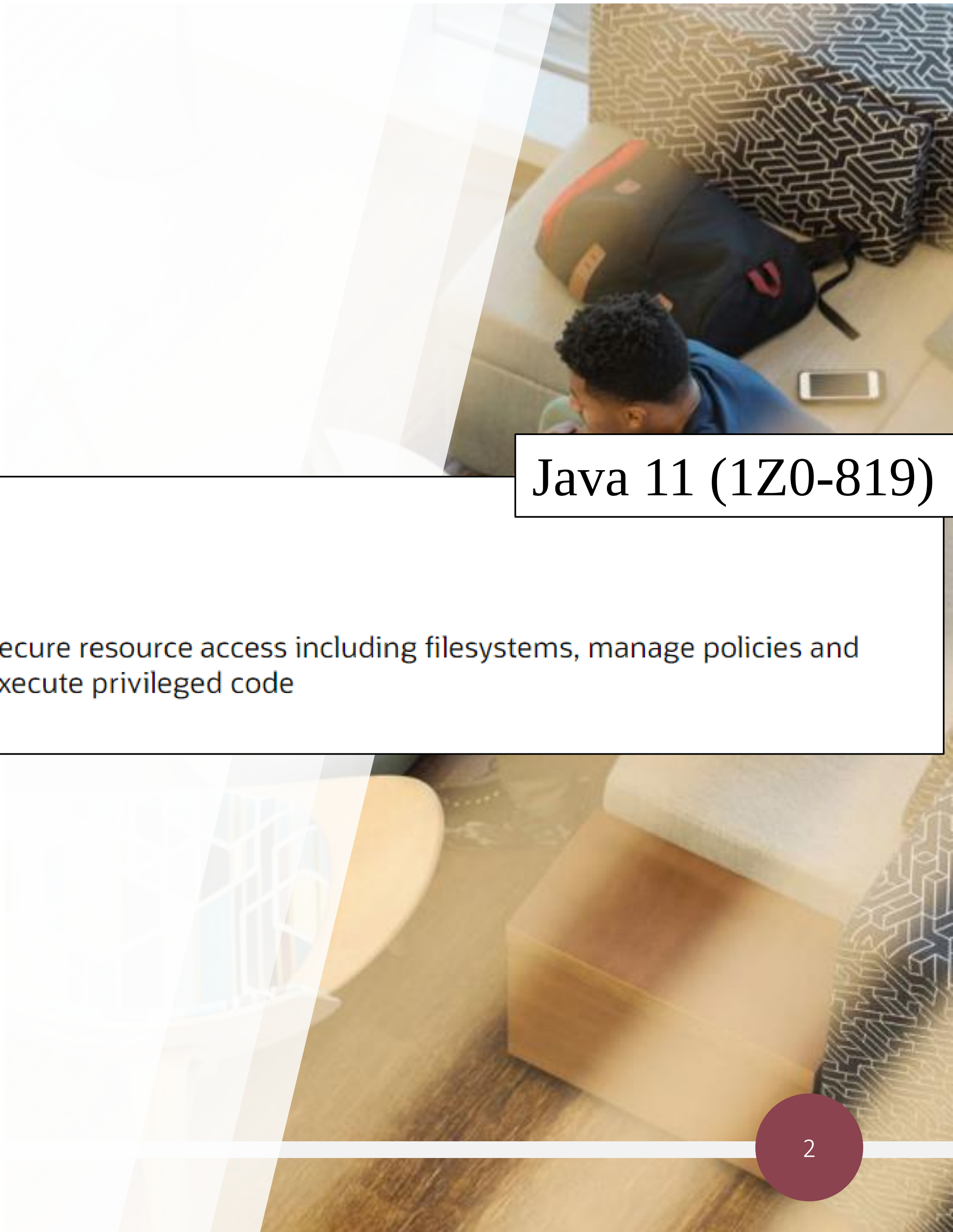
Security

Security



Java 11 (1Z0-819)

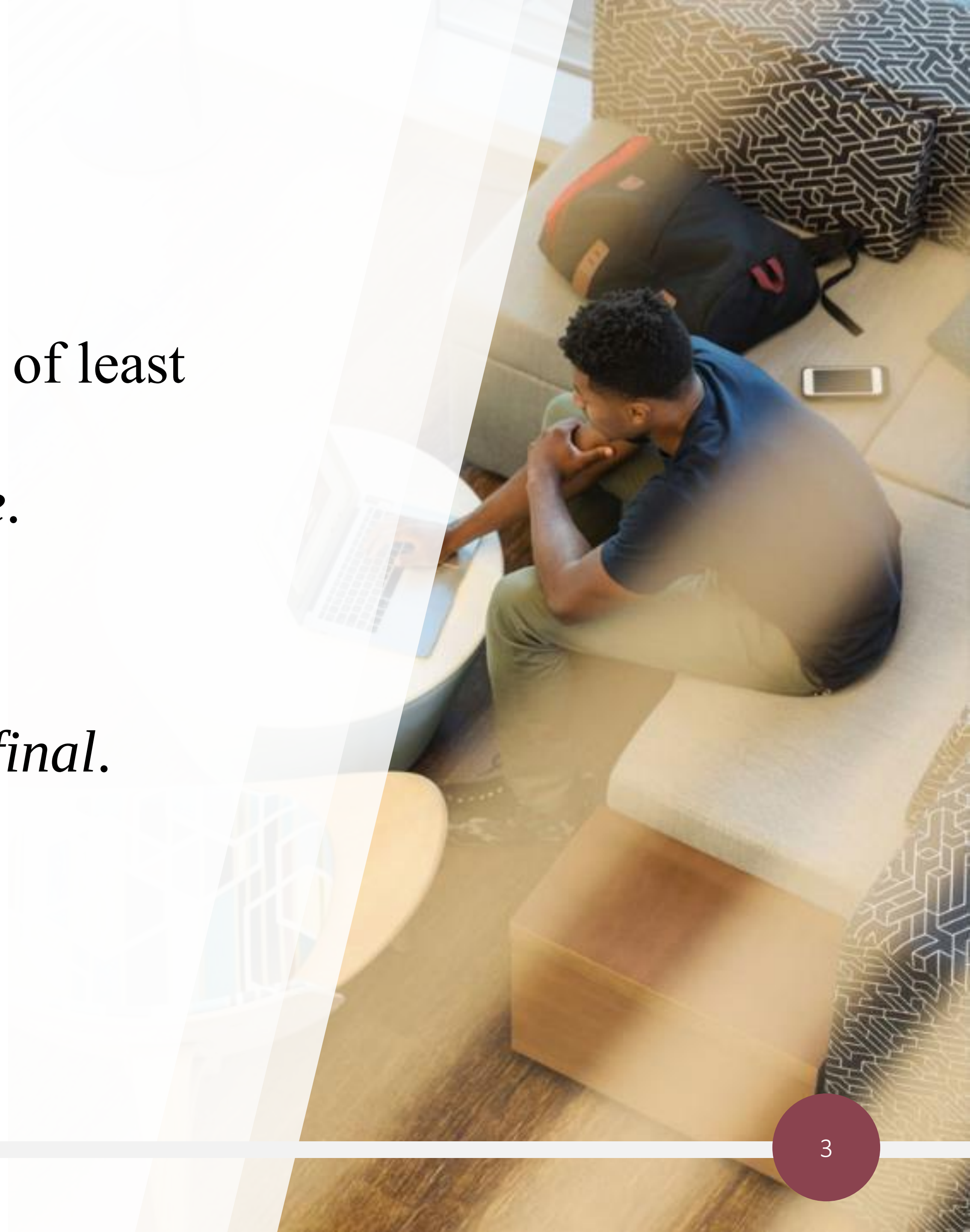
Secure Coding in Java SE Application

- ✓ Develop code that mitigates security threats such as denial of service, code injection, input validation and ensure data integrity
- ✓ Secure resource access including filesystems, manage policies and execute privileged code

Ensuring Data Integrity

- Accessibility
  - limit access as much as possible – “principle of least privilege”.
  - instance variables/methods should be *private*.

- Restrict extensibility
  - prevent subclassing by marking the class as *final*.



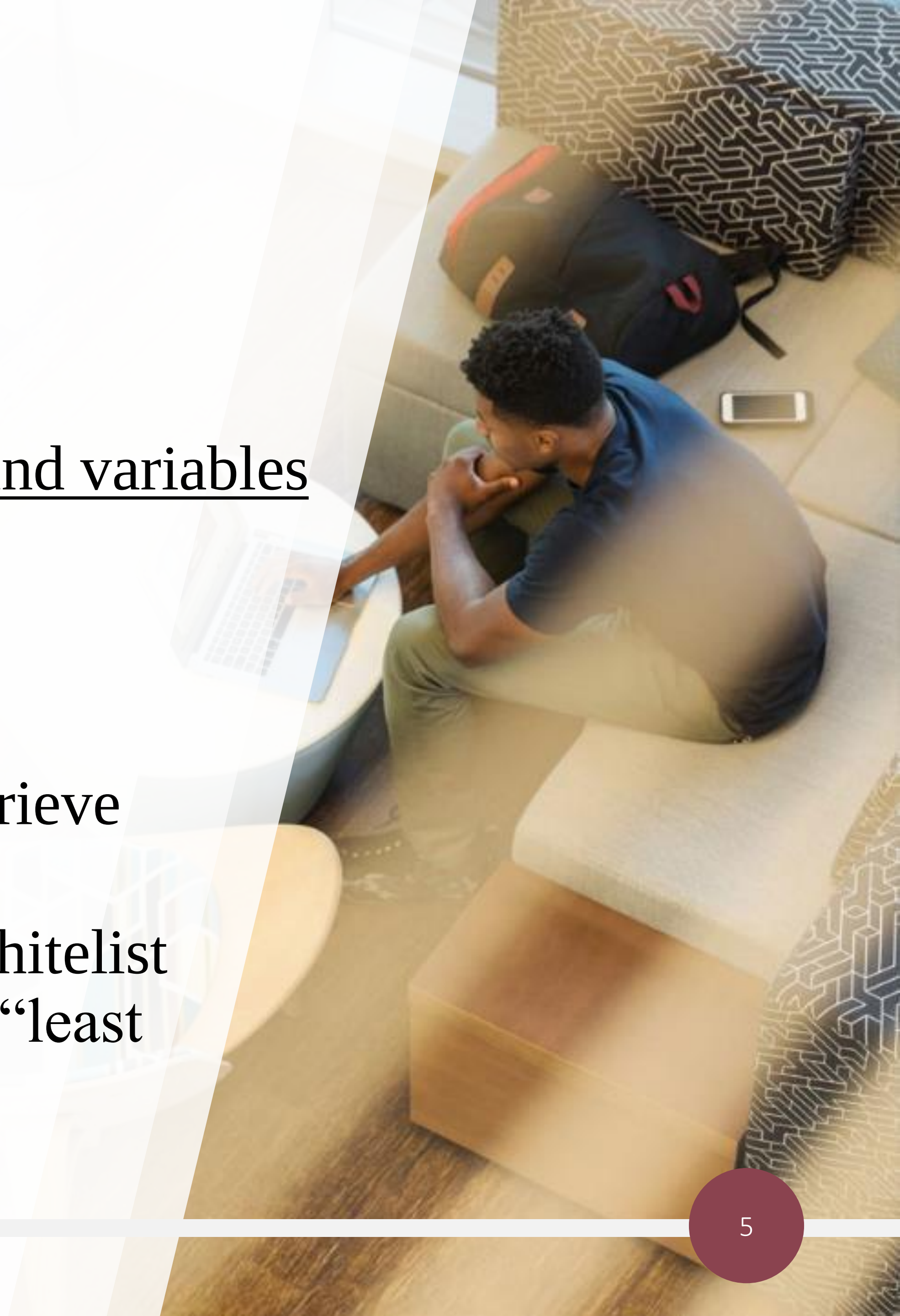
Ensuring Data Integrity

- Immutable objects are objects that cannot be changed after creation.
- They are secure objects and use the following guidelines:
  1. Do not provide any “setter” methods.
  2. Make all the fields *private* and *final*.
  3. Prevent subclassing (prevents overriding):
     a) make the class *final*
     b) make the constructor *private* and provide a *public static* factory method e.g. “createNewInstance”
  4. Instance fields:
     a) immutable types e.g. *String*, ok
     b) mutable types e.g. *StringBuilder*, do NOT share references i.e. use “defensive copying” and “advanced encapsulation”

Injection Attacks

- SQL Injection
  - where user input retrieves unexpected results.
  - protection provided by *PreparedStatement* with bind variables (and not *Statement*).

- Command Injection
  - where operating system commands are used to retrieve unexpected results.
  - protection provided via input validation using a whitelist and/or security policies (applying the principle of "least privilege").
    - applying together provides "defence in depth".



Injection Attacks

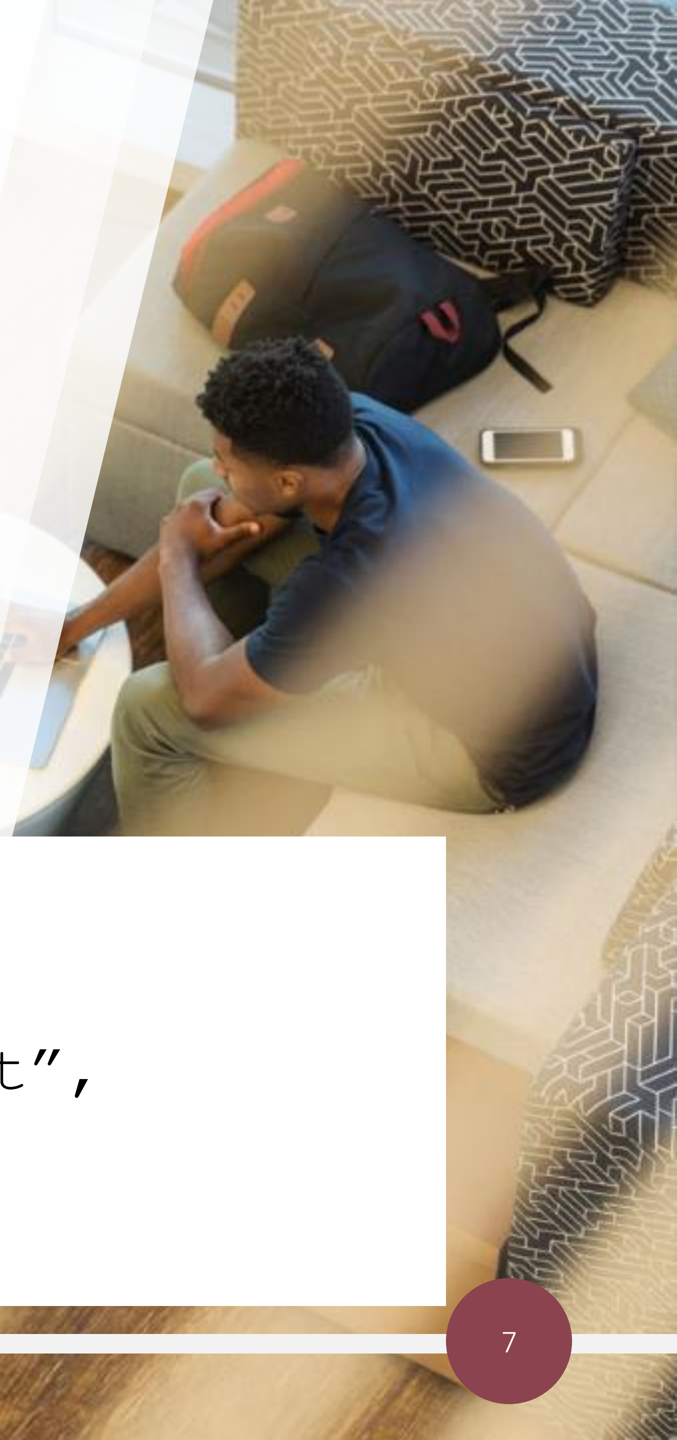
- BankService.java (Netbeans 8)
- CommandInjectionAttack.java

Security Policies

- Can be used in addition to or instead of, a whitelist to prevent command injection attacks.

- If both are applied then this creates a layered approach called “defence in depth”.

```
grant{
     permission java.io.FilePermission
       “c:\\The Farm\\HR\\Staff\\Joe Bloggs.txt”,
        “read”;
};
```



Security Policies

```
grant{
     permission java.io.FilePermission
       “c:\\The Farm\\HR\\Staff\\Joe Bloggs.txt”,
        “read, write”;
};
```

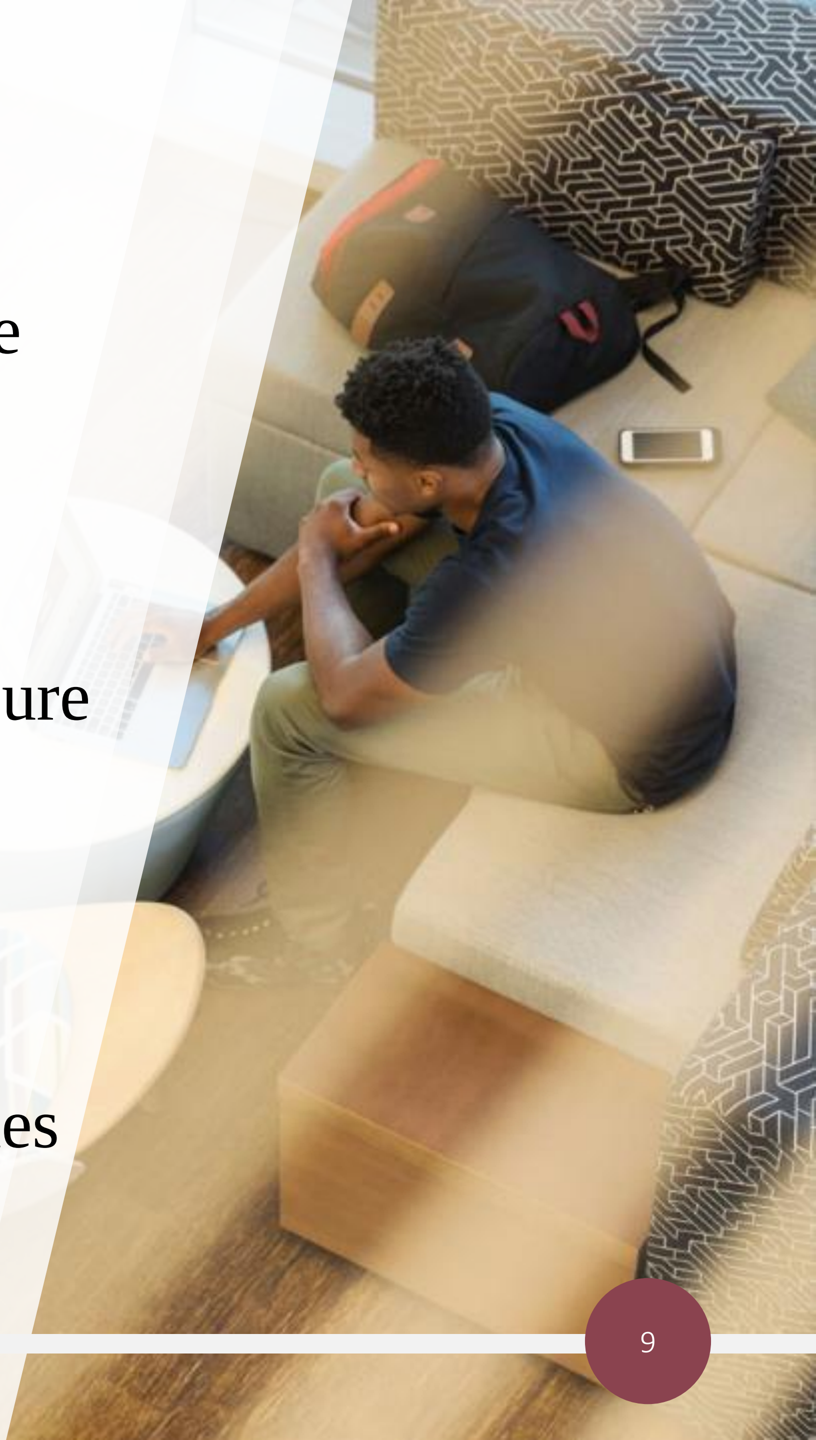
- Be careful that the policy obeys the “principle of least privilege”.

- In other words, if your program only needs to *read* a file, then it should only have *read* permission and not have *write* access.

Denial of Service (DoS) Attacks

- A denial of service attack is where one or more requests are made with the purpose of disrupting services.

- This can be accomplished in a number of ways:
  - leaking resources – always use try-with-resources to ensure you do not leak resources (by not closing them).
  - working with extremely large files – check the file size.
  - inclusion attacks – where a file contains several other files e.g. “zip bomb”.



Guidelines for Confidential Information

- Confidential information includes passwords and personal details such as address, date of birth, salary and account balance.

- Obviously, sensitive data should never be output to the screen, logged or end up in an exception stack trace.

- Data in memory must be protected also:
  - use *char[]* instead of *String*’s.
  - set confidential object references to *null* as soon as you are done with it – makes it immediately eligible for garbage collection.