Web 3

Lesson 5: Hashing

EXAM QUESTIONS…

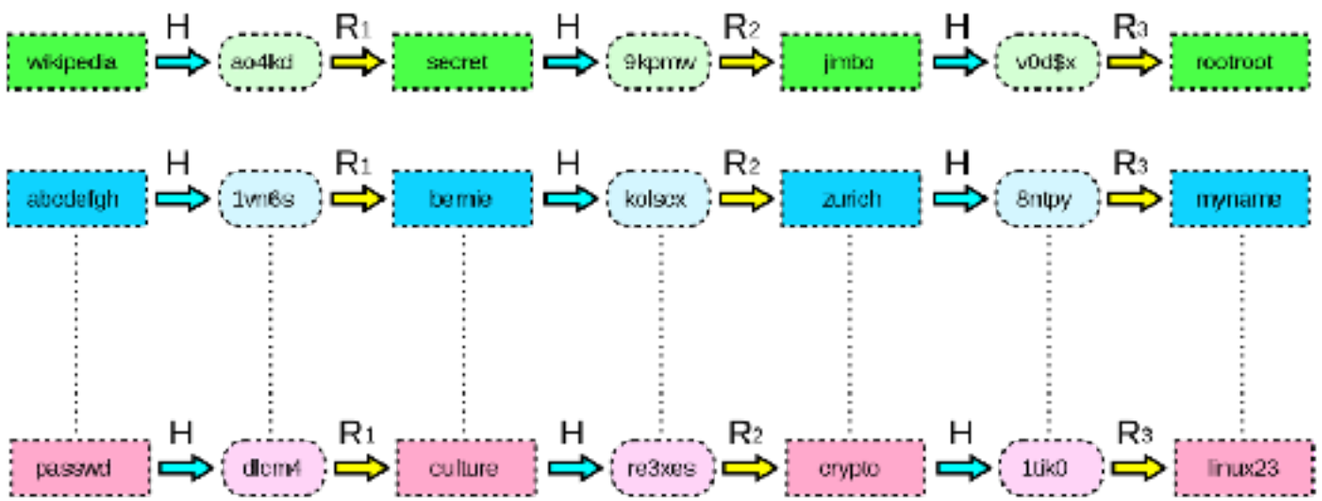
- ☑ Explain why we need to hash our passwords?
- ☑ What is the reason to add a salt before hashing a password?
- ☑ …

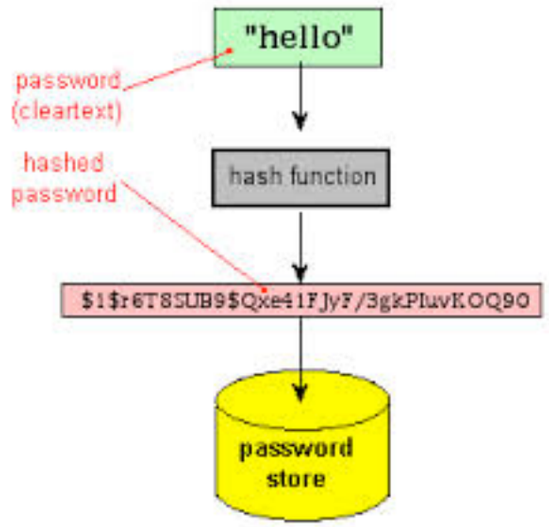
Agenda

- ☐ Hashing
- ☐ Password hashing
- ☐ Salt

HASHING

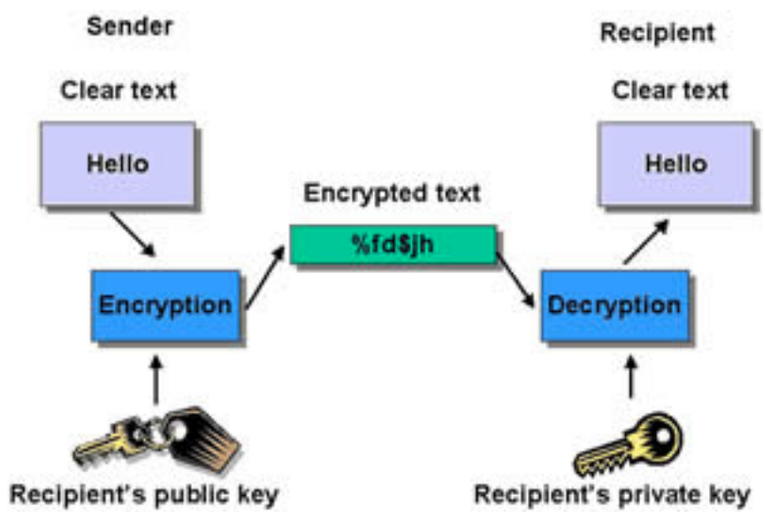
- **Don't** put the passwords plain text in the DB :-)
- **Do** use a hash function
  - MD5
    - unsafe
    - rainbow tables
  - SHA-512
    - safer for todays encryption





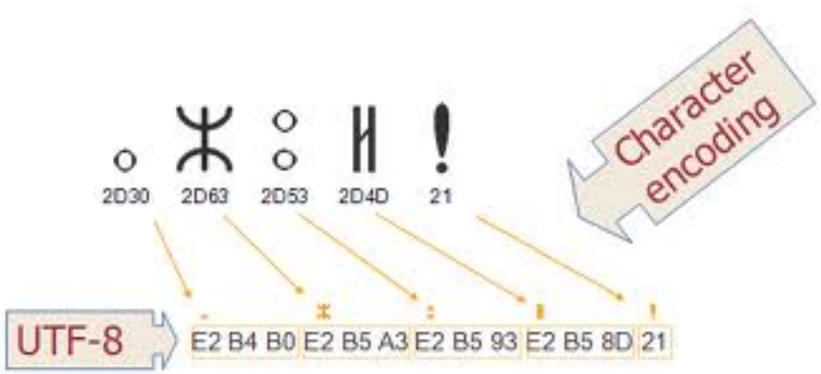
Hashing:

- one-way
- using a salt



Encryption:

- reversible
- using a key



Encoding:

- reversible
- integrity instead of security

SHA-512

- String ⟶ hexadecimal number
  - 128 digits long
  - Example:

    `sha512(banana)` ⟶

    `F8E3183D38E6C51889582CB260AB8252`
    `52F395B4AC8FB0E6B13E9A71F7C10A80`
    `D5301E4A949F2783CB0C20205F1D850F`
    `87045F4420AD2271C8FD5F0CD8944BE3`

SHA-512

- References
  - https://en.wikipedia.org/wiki/Secure_Hash_Algorithms
  - http://passwordsgenerator.net/sha512-hash-generator/

- Hash passwords before storing them in the database

Agenda

- [x] Hashing
- [ ] Password hashing
- [ ] Salt

MESSAGEDIGEST

JAVA.SECURITY

- MessageDigest. getInstance(algorithm)
- update(stringToEncrypt) → encrypt
- digest():byte[ ] → finalize (padding, …)
- reset() → … for further use

https://docs.oracle.com/javase/8/docs/api/java/security/MessageDigest.html

Code Example

```
import java.io.UnsupportedEncodingException;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class HashingExample {

   public static void main(String arg[]) throws Exception {
       System.out.println(sha512("banana"));
     }

   private static String sha512(String password) throws
           NoSuchAlgorithmException, UnsupportedEncodingException {
      //TODO
   }
}
```

Code Example

```
private static String sha512(String password) throws
        NoSuchAlgorithmException, UnsupportedEncodingException {
    MessageDigest crypt = MessageDigest.getInstance("SHA-512");
    crypt.reset();

    byte[] passwordBytes = password.getBytes("UTF-8");
    crypt.update(passwordBytes);

    byte[] digest = crypt.digest();

    return new BigInteger(1, digest).toString(16);
}
```

always use same encoding in any machine!

returns the String representation of this BigInteger as a hexadecimal String

Encoding

String to byte array

- The `getBytes()` method encodes a given String into a sequence of bytes and returns an array of bytes.
  - `password.getBytes()` will use the default encoding for the machine, so you risk a different result on different machines.
  - `password.getBytes("UTF-8")` will always use the UTF-8 charset on any machine, so the result will always be the same.

Encoding

Byte array to **hexadecimal** String

1. Create `BigInteger`:

```
digestBI = new BigInteger(1, digest)
```

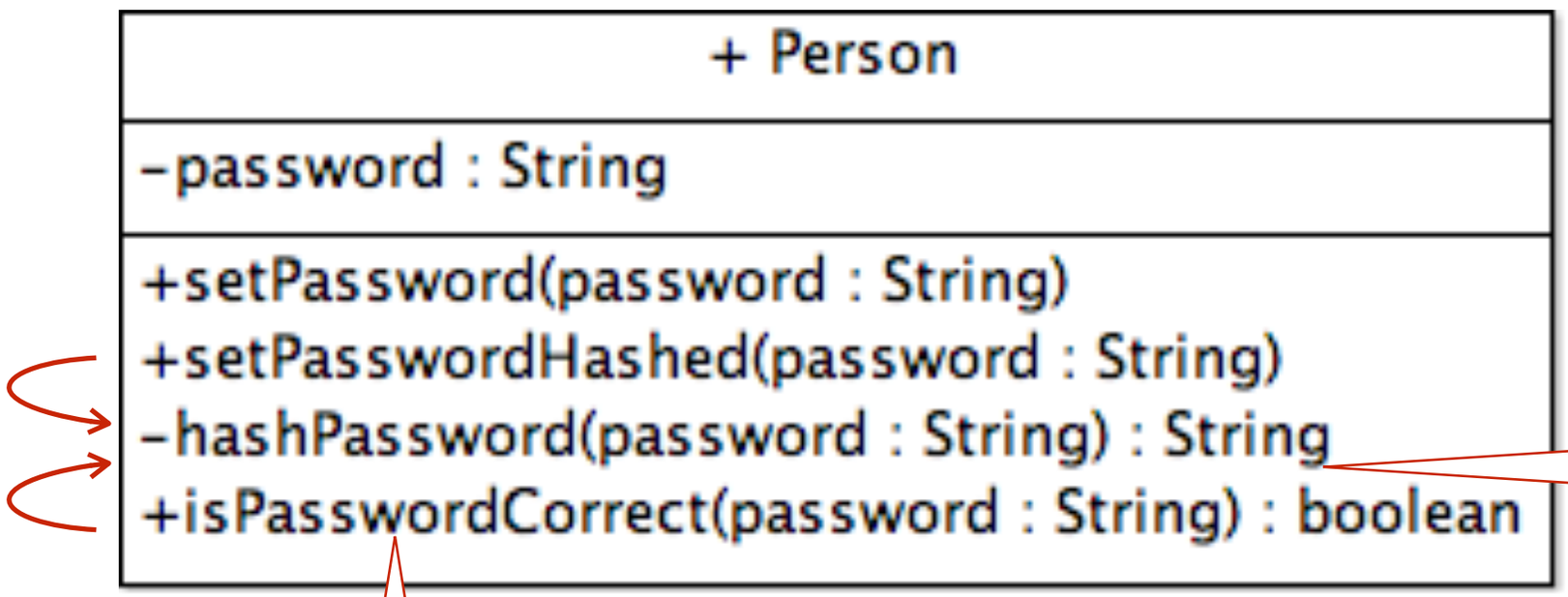
positive

2. Convert `BigInteger` to `String`:

```
digestBI.toString(16)
```

hexadecimal

REAL LIFE EXAMPLE



hash password:
1. create MessageDigest
2. reset
3. update
4. digest
5. convert to String
6. return hashed password

check if password is correct:
1. hash password parameter
2. compare result with (hashed) password property
3. if equal
    return true
   else
    return false

Warning: only throw DomainException here!

Hash Password - v1

```
public class Person {
   private String password;
   …

   private String hashPassword(String password) {
      //create MessageDigest
      MessageDigest crypt = MessageDigest.getInstance("SHA-512");
      //reset
      crypt.reset();
      //update
      byte[] passwordBytes = password.getBytes("UTF-8");
      crypt.update(passwordBytes);
      //digest
      byte[] digest = crypt.digest();
      //convert to String
      BigInteger digestAsBigInteger = new BigInteger(1, digest);
      //return hashed password
      return digestAsBigInteger.toString(16);
   }
}
```

SHA-512 AND DB

- column of password
  - fixed length
  - max 128

Agenda

- [x] Hashing
- [x] Password hashing
- [ ] Salt

USING A SALT

- appending or prepending a random string, called a **salt**, to the password before hashing
  - sha1(salt + password)
  - save both salt and hash in the DB

SECURERANDOM

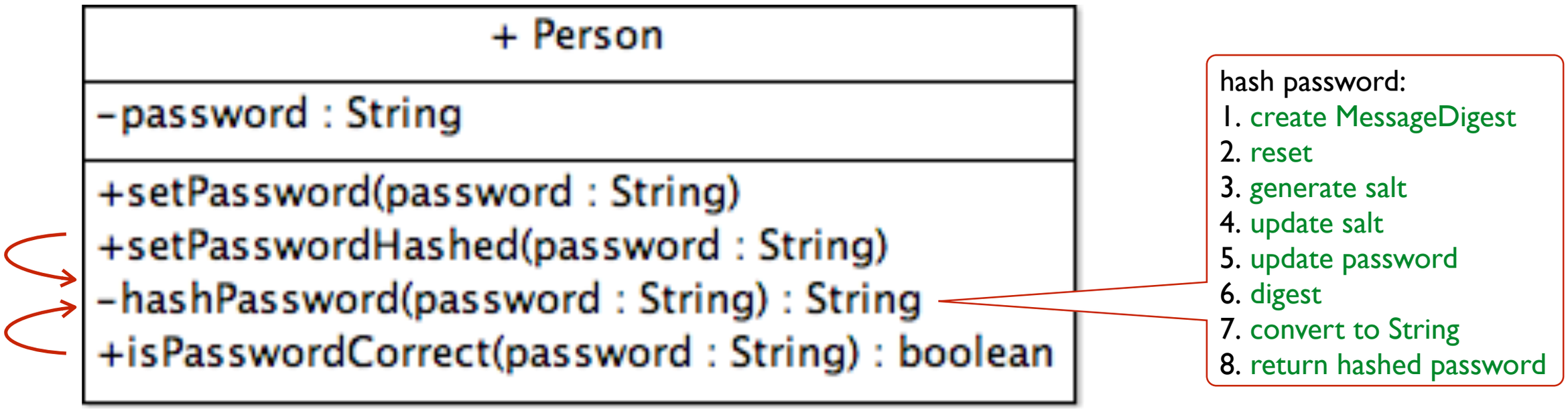
JAVA.SECURITY

- new SecureRandom();
- random.generateSeed(20)

generate seed of 20 bytes

https://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html

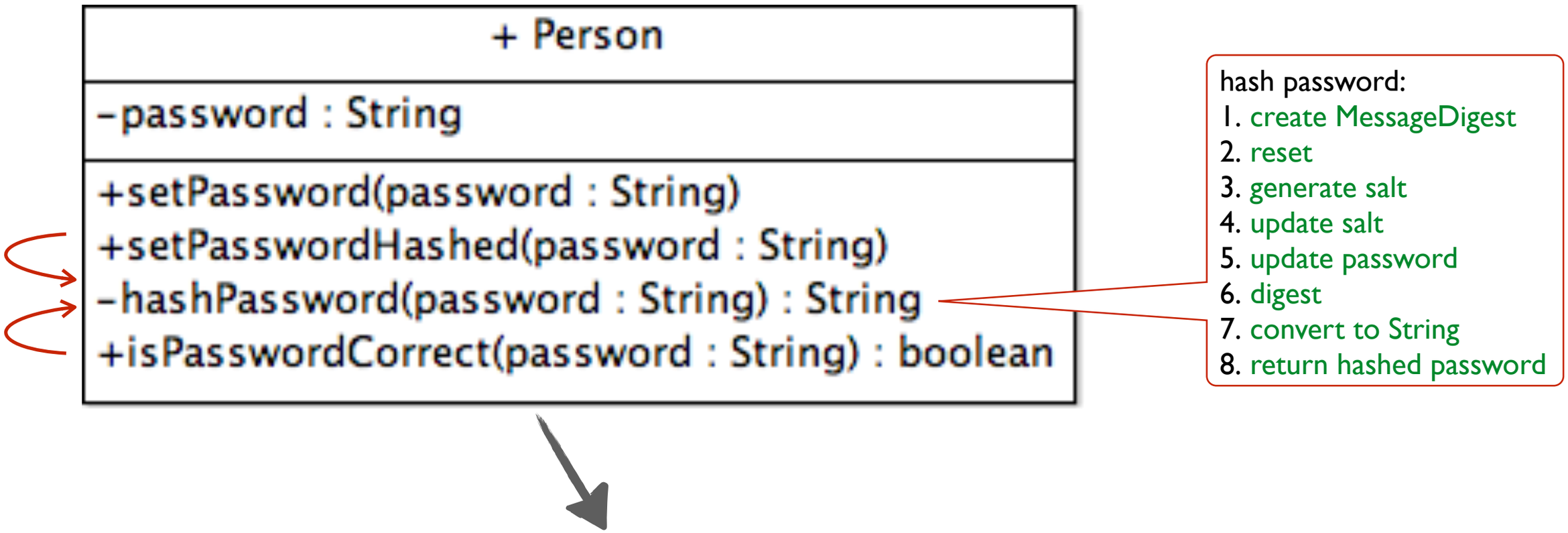
EXAMPLE: PROBLEM



Hash Password - v2

```
private String hashPassword(String password) {
    //create MessageDigest
    MessageDigest crypt = MessageDigest.getInstance("SHA-512");
    //reset
    crypt.reset();
    //create SecureRandom
    SecureRandom random = new SecureRandom();
    //generate seed
    byte[] seed = random.generateSeed(20);
    //update seed
    crypt.update(seed.getBytes("UTF-8"));
    //update password
    byte[] passwordBytes = password.getBytes("UTF-8");
    crypt.update(passwordBytes);
    //digest
    byte[] digest = crypt.digest();
    //convert to String
    BigInteger digestAsBigInteger = new BigInteger(1, digest);
    //return hashed password
    return digestAsBigInteger.toString(16);
}
```

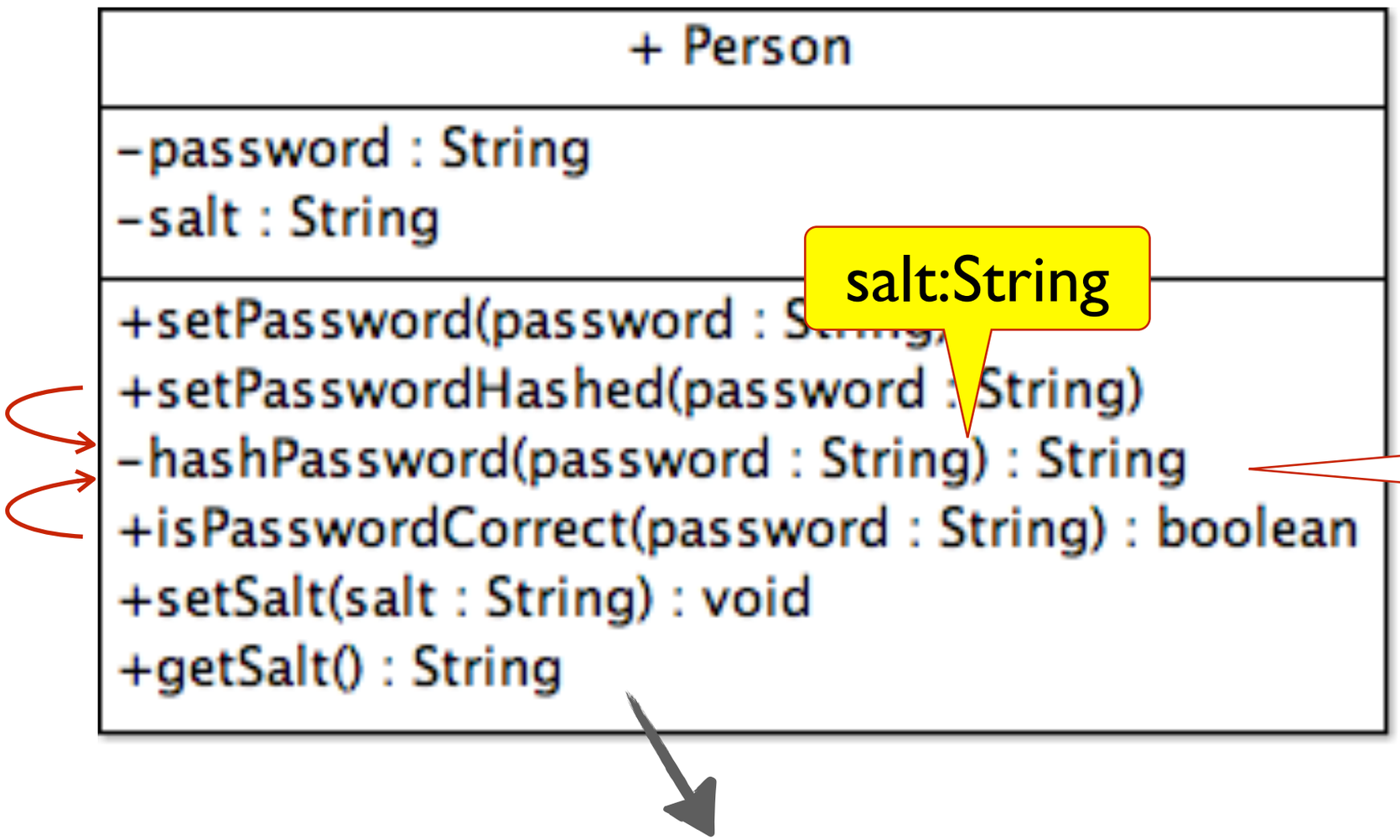
EXAMPLE: PROBLEM



using a random seed generates random hashes

if you use it here,
you will not be able to compare the parameter and the property!

EXAMPLE: SOLUTION



hash password:
1. create MessageDigest
2. reset
3. ~~generate salt~~
4. update salt
5. update password
6. digest
7. convert to String
8. return hashed password

store salt used for password encryption

Agenda

- ☑ Hashing
- ☑ Password hashing
- ☑ Salt

REMARK

- This is not an advanced security class
- Better algorithms might exist
- And more extra Java libraries:
  - Apache Commons Codec
  - …
- You're welcome to try 

REFERENCES

- http://www.differencebetween.info/difference-between-encryption-encoding-and-hashing