Worksheet: Java Generics and Reflection (introspection)

This worksheet reinforces your existing knowledge of Java Generics and Reflection. These *techniques* are commonly used in the technologies we will be examining in the rest of the module.

Required:

You should ensure you are using (at least) version 7 of the JDK.

Preparation:

Create the following `Storage`, `BankAccount` and `Driver` classes in a directory of your choice.

```
class Storage<T> {
    T x;

    public void setValue(T value) {
        x = value;
    }

    public T getValue() {
        return x;
    }
}
```

```
class BankAccount {
    private float balance;

    public void deposit(float amount) {
        this.balance += amount;
    }

    public float showBalance() {
        return this.balance;
    }

    BankAccount() {
        balance = 100;
    }
}
```

```
public class Driver {
    public static void main(String[] args) {
        // YOUR CODE GOES HERE
    }
}
```

Add the following code snippet to your `Driver` class `main` method, creating two different storage objects with two different type specialisations:

```
Storage<BankAccount>  aStorage = new Storage<>();
Storage<String>       sStorage = new Storage<>();
```

1. What are the reasons for using generics here?
2. What are the benefits?
3. Now add the following code to your `Driver` class:

   ```
   Class baCls = BankAccount.class;
   try {
     Object myAccount =  baCls.newInstance();
     aStorage.setValue(myAccount);

     // Deposit
     myAccount.deposit(15);
   }
   catch ( InstantiationException e ) {
     // ...
   }
   catch ( IllegalAccessException e ) {
     // ...
   }
   ```

   Compile and analyse the compiler output.
   What is the cause of the problem reported by the compiler, if any?

4. Now replace:

   ```
   Object myAccount =  baCls.newInstance();
   ```

   with

   ```
   BankAccount myAccount =  baCls.newInstance();
   ```

   How does this affect the compilation process?
   What is the problem, if any?
   What does the `myAccount` variable hold when the code is executed?
   Decide whether your diagnosis from question (3) was correct.

5. Now add an explicit dynamic cast:

   ```
   BankAccount myAccount =  (BankAccount) baCls.newInstance();
   ```

   What does the dynamic cast do here?
   Is it the compiler that performs the cast operation or the Java runtime environment (JVM)?
   Is this code safe?

6. Now replace your initial declaration:

   ```
   Class baCls = BankAccount.class;
   ```

   with

```
Class<BankAccount> baCls = BankAccount.class;
```

Explain the compiler output?
Are there errors?
What is the reason?
What does it say about the role of generics?

7. Now add:

```
System.out.println( aStorage.getValue().showBalance() );

if( aStorage.getClass() == sStorage.getClass() ) {
  System.out.println( "EQUAL" );
} else {
  System.out.println( "NOT EQUAL" );
}
```

What is the run-time output?
Explain why you get such output and how does this relate to generics and their use with reflective instantiation of objects?