```
/*
**  Follow the exercise and answer the questions below.
**  Make sure you have the prog/jdk/5.0 module (or later) loaded.
**  To load the module use: module list || avail || unload || load
**/
```

Generics and read-access reflection (introspection)

Create the following Storage, BankAccount and Main classes in your work directory:

```java
// Storage.java
class Storage<T> {

	T x;

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

	public T getValue() {
		return x;
	}

}


// BankAccount.java
class BankAccount {

	private float saldo;

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

	public float showSaldo() {
		return this.saldo;
	}

	BankAccount() {
		saldo = 100;
	}

}


// Main.java
public class Main {

	 public static void main(String[] args) {
		/** Your code here */
	 }
}
```

Add the following code to your Main program creating two different storage objects with two different type specializations:

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

Q1: What are the reasons of using generics here? What are the benefits? Discuss with your partner and elaborate.

Now add the following code to your Main class

```
Class baCls = BankAccount.class;

try {
       Object myAccount =  baCls.newInstance();
       aStorage.setValue( myAccount );

       // Deposit
       myAccount.deposit( 15 );

}
catch ( InstantiationException e ) {
}
catch ( IllegalAccessException e ) {
}
```

Q2: Compile and analyze compiler output. Given that “everything is an Object” in Java what is the cause of the problem reported by the compiler, if any?

Now replace

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

with

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

Q3: How does this affect the compilation process? What is the problem, if any? What does the myAccount variable hold when the code is executed? Discuss with your partner whether your diagnosis in Q2 was correct.

Now add an explicit dynamic cast

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

Q4: 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*?

*Hint: Try reifying some other meta-level class into `baCls`, e.g., replace

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

with

```
Class baCls = String.class;
```

and compare compiler output.

Now replace your initial declaration

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

with

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

Q5: Explain the compiler output? Are there errors? What is the reason? What does it say about the role of generics?

Now add

```
 //Do a simple output
System.out.println( aStorage.getValue().showSaldo() );

// Now try to compare
if( aStorage.getClass() == sStorage.getClass() ) {
       System.out.println( "EQUAL" );
} else {
       System.out.println( "NOT EQUAL" );
}
```

Q6: What is the run-time output? Reason why you get such an output and how does this relate to generics and their use with reflective instantiation of objects.