```
import java.util.Hashtable;
import java.util.Stack;

public class Transaction implements ITransaction {

	protected IDatabase database = null;
	private ICommandProcessor commandProcessor;
	private IDatabaseData data;
	protected Stack<ICommand> transactionStack;
	protected boolean transactionDone;

	public Transaction(IDatabase database, ICommandProcessor
processor, IDatabaseData data) {
		this.database = database;
		this.commandProcessor = processor;
		this.data = data;
		this.transactionStack = new Stack<>();
		this.commandProcessor.startTransaction();
		this.transactionDone = false;
	}

	@Override
	public Object get(String key) throws
TransactionNotValidException {
		if (!this.data().containsKey(key))
			throw new IllegalArgumentException("Key not
found");
		if (transactionDone)
			throw new
TransactionNotValidException("transaction is not valid");
		return this.data().get(key);
	}

	@Override
	public ITransaction put(String key, Integer value) throws
TransactionNotValidException {
		putHelper(key, value);
		return this;
	}

	@Override
	public ITransaction put(String key, Double value) throws
TransactionNotValidException {
		putHelper(key, value);
		return this;
	}

	@Override
	public ITransaction put(String key, String value) throws
TransactionNotValidException {
		putHelper(key, value);
		return this;
```

```
	}

	@Override
	public ITransaction put(String key, ArrayFormat value)
throws TransactionNotValidException {
		putHelper(key, value);
		return this;
	}

	@Override
	public ITransaction put(String key, ObjectFormat value)
throws TransactionNotValidException {
		putHelper(key, value);
		return this;
	}

	private ITransaction putHelper(String key, Object value)
throws TransactionNotValidException {
		if (value == null)
			throw new IllegalArgumentException("Value cannot
be null");
		if (transactionDone)
			throw new
TransactionNotValidException("transaction is not valid");
		this.commandProcessor.commit(new
AddDataCommand(this.database, key, value), transactionStack);
		return this;
	}

	@Override
	public int getInt(String key) throws
DataTypeMisMatchException, TransactionNotValidException {
		if (!this.data().containsKey(key))
			throw new IllegalArgumentException("Key not
found");
		if (transactionDone)
			throw new
TransactionNotValidException("transaction is not valid");
		int value;
		try {
			value = (int) this.data().get(key);
		} catch (ClassCastException e) {
			throw new DataTypeMisMatchException("value is not
of int type");
		}
		return value;
	}

	@Override
	public double getDouble(String key) throws
DataTypeMisMatchException, TransactionNotValidException {
		if (!this.data().containsKey(key))
```

```
                throw new IllegalArgumentException("Key not
found");
          if (transactionDone)
                throw new
TransactionNotValidException("transaction is not valid");
          double value;
          try {
                value = (double) this.data().get(key);
          } catch (ClassCastException e) {
                throw new DataTypeMisMatchException("value is not
of double type");
          }
          return value;
     }

     @Override
     public ArrayFormat getArray(String key) throws
DataTypeMisMatchException, TransactionNotValidException {
          if (!this.data().containsKey(key))
                throw new IllegalArgumentException("Key not
found");
          if (transactionDone)
                throw new
TransactionNotValidException("transaction is not valid");
          ArrayFormat value = null;
          try {
                value = (ArrayFormat) this.data().get(key);
          } catch (ClassCastException e) {
                throw new DataTypeMisMatchException("value is not
of array type");
          }
          return value;
     }

     @Override
     public String getString(String key) throws
DataTypeMisMatchException, TransactionNotValidException {
          if (!this.data().containsKey(key))
                throw new IllegalArgumentException("Key not
found");
          if (transactionDone)
                throw new
TransactionNotValidException("transaction is not valid");
          String value;
          try {
                value = (String) this.data().get(key);
          } catch (ClassCastException e) {
                throw new DataTypeMisMatchException("value is not
of string type");
          }
          return value;
     }
```

```
      @Override
      public ObjectFormat getObject(String key) throws
DataTypeMisMatchException, TransactionNotValidException {
            if (!this.data().containsKey(key))
                  throw new IllegalArgumentException("Key not
found");
            if (transactionDone)
                  throw new
TransactionNotValidException("transaction is not valid");
            ObjectFormat value = null;
            try {
                  value = (ObjectFormat) this.data().get(key);
            } catch (ClassCastException e) {
                  throw new DataTypeMisMatchException("value is not
of type object");
            }
            return value;
      }

      @Override
      public Object delete(String key) throws
TransactionNotValidException {
            if (transactionDone)
                  throw new
TransactionNotValidException("transaction is not valid");
            Object valueToDelete = this.data.get(key);
            this.commandProcessor.commit(new
DeleteDataCommand(this.database, key, valueToDelete),
transactionStack);
            return valueToDelete;
      }

      @Override
      public boolean modify(String key, Integer value) throws
TransactionNotValidException {
            return modifyHelper(key, value);

      }

      @Override
      public boolean modify(String key, Double value) throws
TransactionNotValidException {
            return modifyHelper(key, value);

      }

      @Override
      public boolean modify(String key, String value) throws
TransactionNotValidException {
            return modifyHelper(key, value);
```

```
	}

	@Override
	public boolean modify(String key, ArrayFormat value) throws
TransactionNotValidException {
		return modifyHelper(key, value);

	}

	@Override
	public boolean modify(String key, ObjectFormat value) throws
TransactionNotValidException {
		return modifyHelper(key, value);

	}

	private boolean modifyHelper(String key, Object data) throws
TransactionNotValidException {
		if (!this.data().containsKey(key))
			throw new IllegalArgumentException("Key not
found");
		if (data == null)
			throw new IllegalArgumentException("Value cannot
be null");
		if (transactionDone)
			throw new
TransactionNotValidException("transaction is not valid");
		return this.commandProcessor.commit(new
ModifyDataCommand(this.database, key, data), transactionStack);
	}

	@SuppressWarnings("unchecked")
	@Override
	public Hashtable<String, Object> data() {
		return (Hashtable<String, Object>) data;
	}

	@Override
	public boolean commit() throws TransactionNotValidException
{
		if (transactionDone)
			throw new
TransactionNotValidException("transaction is not valid");
		this.transactionDone = true;
		if (this.commandProcessor.isTransactionSuccessful()) {

	this.commandProcessor.endTransaction(this.transactionStack);
			return true;
		} else {

	this.commandProcessor.rollbackTransaction(this.transactionSt
ack);
```

```
                return false;
            }
        }

        @Override
        public boolean abort() throws TransactionNotValidException {
            if (transactionDone)
                throw new
TransactionNotValidException("transaction is not valid");
            this.transactionDone = true;

        this.commandProcessor.rollbackTransaction(this.transactionSt
ack);
            return true;
        }

        @Override
        public boolean isActive() throws
TransactionNotValidException {
            if (transactionDone)
                throw new
TransactionNotValidException("transaction is not valid");
            return (!this.transactionStack.isEmpty());
        }

}
```