```
    1: package sdns.serialization;
    2:
    3: import java.io.IOException;
    4: import java.io.InputStream;
    5: import java.io.OutputStream;
    6: import java.nio.ByteBuffer;
    7: import java.nio.ByteOrder;
    8: import java.util.ArrayList;
    9: import java.util.Collections;
   10: import java.util.Objects;
   11:
   12: /**
   13:  * Represents generic SDNS RR and provides serialization/deserialization You
may make concrete anything listed as
   14:  * abstract in this interface. In other words, abstract is not part of the
requirement
   15:  * (while the class, method, and parameters are required).
   16:  *   Credit: Dr. Donahoo of Baylor University for comments and API
   17:  * @version 1.0
   18:  */
   19: public abstract class ResourceRecord {
   20:     protected static final long CN_TYPE_VALUE = 5L;
   21:     protected static final long NS_TYPE_VALUE = 2L;
   22:     protected static final int DOMAIN_NAME_MAX_LEN = 255;
   23:     protected static final int DOMAIN_NAME_LABEL_MAX_LEN = 63;
   24:
   25:     private String name = null;
   26:     private int ttl = -1;
   27:
   28:     protected static boolean isAZaz(char c){
   29:         //I use the -1/+1 syntax because <= makes the assembly check 2
conditions whereas incrementing/decrementing allows
   30:         //  for a faster check (because the +-1 is calculated at compile time)
   31:         return ('A'-1 < c && c < 'Z'+1) || ('a'-1 < c && c < 'z'+1);
   32:     }
   33:
   34:     protected static boolean isAZaz09(char c){
   35:         //I use the -1/+1 syntax because <= makes the assembly check 2
conditions whereas incrementing/decrementing allows
   36:         //  for a faster check (because the +-1 is calculated at compile time)
   37:         return ('0'-1 < c && c < '9'+1) || ('A'-1 < c && c < 'Z'+1) || ('a'-1
< c && c < 'z'+1);
   38:     }
   39:
   40:     protected static boolean isAZaz09Dash(char c){
   41:         //I use the -1/+1 syntax because <= makes the assembly check 2
conditions whereas incrementing/decrementing allows
   42:         //  for a faster check (because the +-1 is calculated at compile time)
   43:         return ('0'-1 < c && c < '9'+1) || ('A'-1 < c && c < 'Z'+1) || ('a'-1
< c && c < 'z'+1) || (c == '-');
   44:     }
   45:
   46:     /**
   47:      * Serializes the given domain name.  First checks that the name is a
valid domain name though...
   48:      * @param name the domain name to validate
   49:      * @param b the byte arraylist to append to
   50:      */
   51:     protected void serializeDomainName(String name, ArrayList<Byte> b) throws
ValidationException {
   52:         if(!this.validateDomainName(name)){
   53:             throw new ValidationException("ERROR: name provided to internal
serialize function not a valid domain name", name);
   54:         }
   55:
   56:         if(name.length() == 1){
   57:             b.add((byte)0);
   58:             return;
   59:         }
   60:
   61:         for(int i=0;i<name.length();i++){
   62:             if((name.charAt(i) == '.' && i != name.length()-1) || i == 0){
   63:                 byte len = 0;
   64:                 //find how long next segment is
   65:                 for(int j=i+1;j<name.length();j++){
   66:                     if(name.charAt(j) == '.'){
   67:                         len = (byte)(j-i-1);
   68:                         break;
   69:                     }
   70:                 }
   71:                 if(i == 0 && name.length() != 1){
   72:                     len++;
   73:                 }
   74:                 b.add(len);
   75:                 if(i == 0){//insert the first letter
   76:                     b.add((byte) name.charAt(0));
   77:                 }
   78:             } else {
   79:                 if(i != name.length()-1){
   80:                     b.add((byte) name.charAt(i));
   81:                 }
   82:             }
   83:         }
   84:
   85:         //Add the final '0' to signal the end of the array
   86:         b.add((byte)0);
   87:     }
   88:
   89:     /**
   90:      * Validates a domain with the following constraints
   91:      * @param domainName the domain name to validate
   92:      * @return whether or not the domain name is valid based on the
Specifications
   93:      */
   94:     protected static boolean validateDomainName(String domainName){
   95:         Objects.requireNonNull(domainName, "Domain names cannot be null");
   96:
   97:         //Name: -- this applies to all domain name fields
   98:         //  Each label must start with a letter, end with a letter or digit,
and have as interior characters only letters
   99:         //    (A-Z and a-z), digits (0-9), and hypen (-).
  100:         //  A name with a single, empty label ("") is acceptable
  101:
  102:         //Base checks
  103:         //A name may not be longer than 255 characters, inclusive of dots
  104:         if(domainName.equals("") || domainName.length() < 1 ||
domainName.length() > DOMAIN_NAME_MAX_LEN){
  105:             return false;
  106:         }
  107:
  108:         //  A name with a single, empty label ("") is acceptable
  109:         if(domainName.equals(".")){
  110:             return true;
  111:         }
  112:
  113:         //Check that the first character isn't an empty label so that when we
split, it doesn't get rid of
  114:         // that bad test case
  115:         //Also check that the last character is a ., which gets lost when
splitting
  116:         if(domainName.charAt(0) == '.' ||
domainName.charAt(domainName.length()-1) != '.'){
  117:             return false;
  118:         }
  119:         //Remove the last dot after validating that it's there because it
causes split to add one extra field
```

```
        domainName = domainName.substring(0, domainName.length()-1);

        //Parse into labels
        String[] labels = domainName.split("\\.", -1);

        if(labels.length < 1){
            return false;
        }

        for(String l : labels){
            //A label may not be longer than 63 characters
            if(l.length() < 1 || l.length() > DOMAIN_NAME_LABEL_MAX_LEN){
                return false;
            }

            //  Each label must start with a letter
            if(!isAZaz(l.charAt(0))){
                return false;
            }

            //  Each label must end with a letter or digit
            if(!isAZaz09(l.charAt(l.length()-1))){
                return false;
            }

            //  Each label must have as interior characters only letters (A-Z and a-z), digits (0-9), and hypen (-)
            for(char c : l.toCharArray()){
                if(!isAZaz09Dash(c)){
                    return false;
                }
            }
        }

        return true;
    }

    private static boolean checkEndOfLabelsBitsSet(byte b){
        return (byte)(b & 0xC0) == (byte)(-64);
    }

    /**
     * Alias for readDomainName(in, -1) <-sentinal value
     * @param in the InputStream to read from
     * @return a string representing the deserialized domain name read
     * @throws ValidationException if parse or validation problem
     * @throws IOException if I/O problem
     */
    private static String readDomainName(InputStream in) throws ValidationException, IOException {
        return readDomainName(in, -1);
    }

    /**
     *
     * @param in the InputStream to read from
     * @param maxSize if the number of bytes read doesn't match the maxSize, a ValidationException is thrown
     * @return a string representing the deserialized domain name read
     * @throws ValidationException if parse or validation problem
     * @throws IOException if I/O problem
     */
    private static String readDomainName(InputStream in, int maxSize) throws ValidationException, IOException {
        Objects.requireNonNull(in, "Input stream cannot be null");

        int numBytes = 0;
        byte llen;
        char ch;
        StringBuilder dname = new StringBuilder();
        String finalString = "";

        //Check if maxSize is 0, in which case throw a validation exception
        if(maxSize == 0){
            throw new ValidationException("ERROR: Max size of a domain name cannot be 0", maxSize + "");
        }

        //read the first length
        llen = (byte) in.read();//discards the top 3 bytes, might output a negative
        numBytes++;

        //read the label length, label values, repeat until you read the end
        while(llen > 0 && !checkEndOfLabelsBitsSet(llen)){//0 is end of labels, -1 is end of stream
            for(int i=0; i<llen; i++){
                //read a character
                ch = (char)((byte) in.read());
                numBytes++;
                dname.append(ch);
            }
            dname.append('.');

            if(dname.length()-1 > DOMAIN_NAME_LABEL_MAX_LEN+1){//-1 for the '.'
                throw new ValidationException("Label cannot exceed " + DOMAIN_NAME_LABEL_MAX_LEN + " characters (including .)", dname.length() + "");
            }

            //spew into the output buffer
            finalString += dname.toString();
            dname.setLength(0);

            llen = (byte) in.read();
            numBytes++;
        }

        if(llen < -1){
            throw new ValidationException("ERROR: label length < 0", llen + "");
        } else if(llen == -1){
            throw new IOException("ERROR: End of input stream");
        }

        if(checkEndOfLabelsBitsSet(llen)){//clear the next byte too, according to the specifications
            in.read();
            numBytes++;
        }

        //Check for max size violations
        if(maxSize > 0 && maxSize != numBytes){
            throw new ValidationException("ERROR: RDLENGTH does not match RDATA length (rdlen=" + maxSize
                    + ", rdata.length()=" + numBytes, maxSize + "");
        }

        if(numBytes == 1){//then we have the '.' case
            finalString += '.';
        }

        return finalString;
    }

    private static int readIntBigEndian(InputStream in) throws IOException {
```

```
244:         int toReturn = 0;
245:         byte temp;
246:
247:         for(int i=0;i<4;i++){
248:             toReturn = toReturn << 8;
249:             temp = (byte) in.read();
250:             toReturn |= (temp & 0x000F);
251:         }
252:
253:         return toReturn;
254:     }
255:
256:     private static int readUnsignedShortBigEndian(InputStream in) throws
IOException {
257:         int toReturn = 0;
258:         byte temp;
259:
260:         for(int i=0;i<2;i++){
261:             toReturn = toReturn << 8;
262:             temp = (byte) in.read();
263:             toReturn |= (temp & 0x00FF);
264:         }
265:
266:         return toReturn;
267:     }
268:
269:     /**
270:      * Deserializes message from input source
271:      * @param in deserialization input source
272:      * @return a specific RR resulting from deserialization
273:      * @throws ValidationException if parse or validation problem
274:      * @throws IOException if I/O problem (e.g., premature EoS)
275:      * @throws NullPointerException if in is null
276:      */
277:     public static ResourceRecord decode(InputStream in) throws
ValidationException, IOException {
278:         Objects.requireNonNull(in, "Input stream cannot be null");
279:
280:         //don't have to do any validation because it's done in all of the
subclasses -- is this ok to do?
281:         //name
282:         String name = readDomainName(in);
283:         //Type
284:         byte temp, type;
285:         temp = (byte) in.read();
286:         type = (byte) in.read();
287:
288:         //validate
289:         if(temp != 0){
290:             throw new ValidationException("ERROR: type bytes not set
correctly: [" + temp + " " + type + "]", temp + "");
291:         }
292:         if(type != CN_TYPE_VALUE && type != NS_TYPE_VALUE){
293:             throw new ValidationException("ERROR: invalid type", type + "");
294:         }
295:
296:         //0x0001
297:         temp = (byte) in.read();
298:         if(temp != 0){
299:             throw new ValidationException("ERROR: 0x0001 first byte not set
correctly", temp + "");
300:         }
301:         temp = (byte) in.read();
302:         if(temp != 1){
303:             throw new ValidationException("ERROR: 0x0001 second byte not set
correctly", temp + "");
304:         }
305:
306:         //TTL
307:         int ttl = readIntBigEndian(in);
308:
309:         //RDLength
310:         int rdlen = readUnsignedShortBigEndian(in);
311:
312:         //RData
313:         String rdata = readDomainName(in, rdlen);
314:
315:         //Build the object (which checks all of the fields
316:         ResourceRecord toReturn;
317:         //can't switch on a long apparently
318:         if(type == CN_TYPE_VALUE){
319:             toReturn = new CName(name, ttl, rdata);
320:         } else if(type == NS_TYPE_VALUE){
321:             toReturn = new NS(name, ttl, rdata);
322:         } else {
323:             throw new ValidationException("ERROR: INTERNAL ERROR, PLEASE
REPORT.  Type=" + type, type + "");
324:         }
325:
326:         return toReturn;
327:     }
328:
329:     /**
330:      * Serializes RR to given sink
331:      * @param out serialization sink
332:      * @throws IOException if I/O problem
333:      * @throws NullPointerException if out is null
334:      */
335:     public void encode(OutputStream out) throws IOException {
336:         Objects.requireNonNull(out, "Output stream cannot be null");
337:
338:         /*
339:             //foo. = 3 102, 111, 111, 192, 5 //-64 signed = 192 unsigned
340:             Valid data:
341:             byte[] buff = { 3, 'f', 'o', 'o', -64, 5,
342:                                     0, 2,
343:                                     0, 1, //0x0001
344:                                     0, 0, 0, 0,
345:                                     6,
346:                                     3, 'f', 'o', 'o', -64, 5};//"foo."
347:          */
348:         //Construct the output
349:         ArrayList<Byte> a = new ArrayList<>();
350:
351:         //Name
352:         try{
353:             this.serializeDomainName(this.getName(), a);
354:         } catch(ValidationException e){
355:             //Do nothing
356: //            throw new ValidationException("WARN WARN WARN: \"NAME\" FIELD
DOES NOT CONTAIN A VALID DOMAIN NAME", this.getName());
357:         }
358:
359:         //Type -- this is not expandable, and is written acknowledging the
"quick and dirty" way was used.
360:         Collections.addAll(a, (byte)0, (byte)this.getTypeValue());
361:
362:         //0x0001
363:         Collections.addAll(a, (byte)0, (byte)1);
364:
365:         //TTL
366:         ByteBuffer buff = ByteBuffer.allocate(4);
367:         buff.order(ByteOrder.BIG_ENDIAN);
368:         buff.putInt(this.getTTL());
369:         Collections.addAll(a, buff.get(0), buff.get(1), buff.get(2),
buff.get(3));
```

```
370:
371:         //RData and RDLength
372:         ArrayList<Byte> rdataBytes = this.serializeRData();
373:         short rdlen = (short) rdataBytes.size();
374:
375:         ByteBuffer buff2 = ByteBuffer.allocate(4);
376:         buff2.order(ByteOrder.BIG_ENDIAN);
377:         buff2.putShort(rdlen);
378:
379:         //RDLength
380:         Collections.addAll(a, buff2.get(0), buff2.get(1));
381:         //RData
382:         a.addAll(rdataBytes);
383:
384:         byte[] finalBuff = new byte[a.size()];
385:         for(int i=0;i<a.size();i++){
386:             finalBuff[i] = a.get(i);
387:         }
388:         out.write(finalBuff);
389:     }
390:
391:     /**
392:      * Return type value for specific RR
393:      * @return type value
394:      */
395:     public abstract long getTypeValue();
396:
397:     /**
398:      * Get name of RR
399:      * @return name
400:      */
401:     public String getName() { return this.name; }
402:
403:     /**
404:      * Set name of RR
405:      * @param name new name of RR
406:      * @return this RR with new name
407:      * @throws ValidationException if new name invalid or null
408:      */
409:     public ResourceRecord setName(String name) throws ValidationException {
410:         try {
411:             Objects.requireNonNull(name, "Name cannot be null");
412:         } catch(NullPointerException n){
413:             throw new ValidationException(n.getMessage(), name);
414:         }
415:
416:         //require non null and validate domain name all in one!
417:         if(this.validateDomainName(name)){
418:             this.name = name;
419:         } else {
420:             throw new ValidationException("Name did not pass domain name
checks", name);
421:         }
422:
423:         return this;
424:     }
425:
426:     /**
427:      * Get TTL of RR
428:      * @return TTL
429:      */
430:     public int getTTL(){ return this.ttl; }
431:
432:     /**
433:      * Set TTL of RR
434:      * @param ttl new TTL
435:      * @return this RR with new TTL
436:      * @throws ValidationException if new TTL invalid
437:      */
438:     public ResourceRecord setTTL(int ttl) throws ValidationException {
439:         if(ttl < 0){
440:             throw new ValidationException("TTL < 0", ttl + "");
441:         }
442:         this.ttl = ttl;
443:         return this;
444:     }
445:
446:     /**
447:      * Returns a byte array of the rdata for this object.  For internal use
only.
448:      * @return the serialized version of this objects rdata
449:      */
450:     protected abstract ArrayList<Byte> serializeRData();
451: }
```

```
    1: package sdns.serialization;
    2:
    3: import java.io.IOException;
    4: import java.io.OutputStream;
    5: import java.nio.ByteBuffer;
    6: import java.nio.ByteOrder;
    7: import java.util.ArrayList;
    8: import java.util.Arrays;
    9: import java.util.Collections;
   10: import java.util.Objects;
   11:
   12: /**
   13:  * Represents a CName and provides serialization/deserialization
   14:  *   Credit: Dr. Donahoo of Baylor University for comments and API
   15:  * @version 1.1
   16:  */
   17: public class CName extends ResourceRecord {
   18:     private String canonicalName;
   19:
   20:     /**
   21:      * Constructs CName using given values
   22:      * @param name RR name
   23:      * @param ttl RR TTL
   24:      * @param canonicalName Canonical name
   25:      * @throws ValidationException if validation fails (see specification),
including null name or canonical name
   26:      */
   27:     public CName(String name, int ttl, String canonicalName) throws
ValidationException {
   28:         this.setTTL(ttl);
   29:         //require non null, domain name validation happens in each individual
method
   30:         try {
   31:             this.setName(Objects.requireNonNull(name, "Name cannot be null"));
   32:         } catch(NullPointerException n){
   33:             throw new ValidationException(n.getMessage(), name);
   34:         }
   35:         try {
   36:             this.setCanonicalName(Objects.requireNonNull(canonicalName,
"Canonical Name cannot be null"));
   37:         } catch(NullPointerException n){
   38:             throw new ValidationException(n.getMessage(), canonicalName);
   39:         }
   40:     }
   41:
   42:     /**
   43:      * Get canonical name
   44:      * @return name
   45:      */
   46:     public String getCanonicalName() { return this.canonicalName; }
   47:
   48:     /**
   49:      * Set canonical name
   50:      * @param canonicalName new canonical name
   51:      * @return this RR with new canonical name
   52:      * @throws ValidationException if invalid canonical name, including null
   53:      */
   54:     public CName setCanonicalName(String canonicalName) throws
ValidationException {
   55:         try {
   56:             Objects.requireNonNull(canonicalName, "Canonical Name cannot be
null");
   57:         } catch(NullPointerException n){
   58:             throw new ValidationException(n.getMessage(), canonicalName);
   59:         }
   60:
   61:         //require non null and validate domain name all in one!
   62:         if(this.validateDomainName(canonicalName)){
   63:             this.canonicalName = canonicalName;
   64:         } else {
   65:             throw new ValidationException("Canonical Name did not pass domain
name checks", canonicalName);
   66:         }
   67:
   68:         return this;
   69:     }
   70:
   71:     /**
   72:      * Returns a String representation
   73:      * CName: name=<name> ttl=<ttl> canonicalname=<canonicalname>
   74:      *   For example
   75:      *     CName: name=foo.com. ttl=500 canonicalname=ns.com
   76:      *
   77:      * @return String representation
   78:      */
   79:     @Override
   80:     public String toString() { return "CName: name=" + this.getName() + "
ttl=" + this.getTTL() + " canonicalname=" + this.getCanonicalName(); }
   81:
   82:     /**
   83:      * Return type value for specific RR
   84:      * @return type value
   85:      */
   86:     @Override
   87:     public long getTypeValue() { return CN_TYPE_VALUE; }
   88:
   89:     /**
   90:      * Returns a byte array of the rdata for this object.  For internal use
only.
   91:      * @return the serialized version of this objects rdata
   92:      */
   93:     @Override
   94:     protected ArrayList<Byte> serializeRData(){
   95:         ArrayList<Byte> rdataBytes = new ArrayList<>();
   96:         try {
   97:             this.serializeDomainName(this.getCanonicalName(), rdataBytes);
   98:         } catch (ValidationException e) {
   99:             //Do nothing
  100: //            throw new ValidationException("WARN WARN WARN:
\"CANONICALNAME\" FIELD DOES NOT CONTAIN A VALID DOMAIN NAME", this.getName());
  101:         }
  102:
  103:         return rdataBytes;
  104:     }
  105: }
```

```
    1: package sdns.serialization;
    2:
    3: //import java.io.Serial;
    4: import java.io.Serializable;
    5:
    6: /**
    7:  * Exception for handling validation problems
    8:  *   Credit: Dr. Donahoo of Baylor University for comments and API
    9:  * @version 1.0
   10:  */
   11: public class ValidationException extends Exception implements Serializable {
   12: //    @Serial
   13:     String badToken;
   14:
   15:     /**
   16:      * Equivalent to ValidationException(message, null, badToken)
   17:      * @param message exception message
   18:      * @param badToken string causing exception (null if no such string)
   19:      */
   20:     public ValidationException(String message, String badToken){
   21:         super(message);
   22:         this.badToken = badToken;
   23:     }
   24:
   25:     /**
   26:      * Constructs validation exception
   27:      * @param message exception message
   28:      * @param cause exception cause
   29:      * @param badToken string causing exception (null if no such string)
   30:      */
   31:     public ValidationException(String message, Throwable cause, String
badToken){
   32:         super(message, cause);
   33:         this.badToken = badToken;
   34:     }
   35:
   36:     /**
   37:      * Returns bad token
   38:      * @return bad token
   39:      */
   40:     public String getBadToken(){ return this.badToken; }
   41: }
```

```
    1: package sdns.serialization;
    2:
    3: import java.io.OutputStream;
    4: import java.util.ArrayList;
    5:
    6: /**
    7:  * Represents an unknown type and provide deserialization
    8:  *   Credit: Dr. Donahoo of Baylor University for comments and API
    9:  * @version 1.1
   10:  */
   11: public class Unknown extends ResourceRecord {
   12:
   13:     /**
   14:      * Always throws UnsupportedOperationException
   15:      * @param out serialization sink
   16:      */
   17:     @Override
   18:     public void encode(OutputStream out) throws UnsupportedOperationException
{ throw new UnsupportedOperationException(); }
   19:
   20:     /**
   21:      * Returns a byte array of the rdata for this object.  For internal use
only.
   22:      * @return the serialized version of this objects rdata
   23:      */
   24:     @Override
   25:     protected ArrayList<Byte> serializeRData() throws
UnsupportedOperationException { throw new UnsupportedOperationException(); }
   26:
   27:     @Override
   28:     public long getTypeValue() { return -1L; }
   29:
   30:     /**
   31:      * Returns a String representation
   32:      * Unknown: name=<name> ttl=<ttl>
   33:      *   For example
   34:      *     Unknown: name=foo.com. ttl=500
   35:      * @return a String representation
   36:      */
   37:     @Override
   38:     public String toString() { return "Unknown: name=" + this.getName() + "
ttl=" + this.getTTL(); }
   39: }
```

```
    1: package sdns.serialization;
    2:
    3: import java.io.IOException;
    4: import java.io.OutputStream;
    5: import java.util.ArrayList;
    6: import java.util.Objects;
    7:
    8: /**
    9:  * Represents a NS and provides serialization/deserialization
   10:  *   Credit: Dr. Donahoo of Baylor University for comments and API
   11:  * @version 1.1
   12:  */
   13: public class NS extends ResourceRecord {
   14:     private String nameServer;
   15:
   16:     /**
   17:      * Constructs NS using given values
   18:      * @param name RR name
   19:      * @param ttl RR TTL
   20:      * @param nameServer name server
   21:      * @throws ValidationException if validation fails (see specification),
including null name or nameServer
   22:      */
   23:     public NS(String name, int ttl, String nameServer) throws
ValidationException {
   24:         this.setTTL(ttl);
   25:         //require non null, domain name validation happens in each individual
method
   26:         try {
   27:             this.setName(Objects.requireNonNull(name, "Name cannot be null"));
   28:         } catch(NullPointerException n){
   29:             throw new ValidationException(n.getMessage(), name);
   30:         }
   31:         try {
   32:             this.setNameServer(Objects.requireNonNull(nameServer, "Name
Server cannot be null"));
   33:         } catch(NullPointerException n){
   34:             throw new ValidationException(n.getMessage(), nameServer);
   35:         }
   36:     }
   37:
   38:     /**
   39:      * Get name server
   40:      * @return name
   41:      */
   42:     public String getNameServer() { return this.nameServer; }
   43:
   44:     /**
   45:      * Set name server
   46:      * @param nameServer new name server
   47:      * @return this NS with new name server
   48:      * @throws ValidationException if invalid name server, including null
   49:      */
   50:     public NS setNameServer(String nameServer) throws ValidationException {
   51:         try {
   52:             Objects.requireNonNull(nameServer, "Name Server cannot be null");
   53:         } catch(NullPointerException n){
   54:             throw new ValidationException(n.getMessage(), nameServer);
   55:         }
   56:
   57:         //require non null and validate domain name all in one!
   58:         if(this.validateDomainName(nameServer)){
   59:             this.nameServer = nameServer;
   60:         } else {
   61:             throw new ValidationException("Name Server did not pass domain
name checks", nameServer);
   62:         }
   63:
   64:         return this;
   65:     }
   66:
   67:     /**
   68:      * Returns a String representation
   69:      * NS: name=<name> ttl=<ttl> nameserver=<nameserver>
   70:      *   For example
   71:      *     NS: name=foo.com. ttl=500 nameserver=ns.com
   72:      * @return a String representation
   73:      */
   74:     @Override
   75:     public String toString() { return "NS: name=" + this.getName() + " ttl="
+ this.getTTL() + " nameserver=" + this.getNameServer(); }
   76:
   77:     @Override
   78:     public long getTypeValue() { return NS_TYPE_VALUE; }
   79:
   80:     /**
   81:      * Returns a byte array of the rdata for this object.  For internal use
only.
   82:      * @return the serialized version of this objects rdata
   83:      */
   84:     @Override
   85:     protected ArrayList<Byte> serializeRData(){
   86:         ArrayList<Byte> rdataBytes = new ArrayList<>();
   87:         try {
   88:             this.serializeDomainName(this.getNameServer(), rdataBytes);
   89:         } catch (ValidationException e) {
   90:             //Do nothing
   91: //            throw new ValidationException("WARN WARN WARN: \"NAMESERVER\"
FIELD DOES NOT CONTAIN A VALID DOMAIN NAME", this.getName());
   92:         }
   93:
   94:         return rdataBytes;
   95:     }
   96: }
```