Bref format specification (version 2)

General information and overview:

1. Bref (pronounced “bee-ref”) stands for “binary reference”. Bref format is a binary format for storing phased, non-missing genotypes for a list of samples.

2. This document provides pseudocode for reading for reading Bref format. The pseudocode defines the structure of a Bref file.

3. Integer values are read using the readByte() and readInt() methods described in the documentation for the Java DataInputStream interface in the java.io package.
    a. The readByte() method reads a signed one-byte integer in the range: [-128, 127].
    b. The readInt() method stores a signed four-byte integer in the range: [-2^{31}, $2^{31} – 1$].

4. String values are read using the readUTF() method described in the documentation for the Java DataInputStream interface in the java.io package.

5. The Bref format stores the genotype data in data blocks. Each data block contains the marker and genotype information for a set of consecutive markers. Each marker is either “index-coded” or “sequence-coded”.
    a. If a marker is index-coded, the indices of haplotypes carrying non-major alleles are stored. This is an efficient storage format for markers whose non-major alleles have low frequency
    b. For markers that are sequence-coded, the set of distinct allele sequences present in the sequence-coded markers in the data block is stored, and the index of the distinct allele sequence carried by each haplotype is stored.
    c. The number of distinct allele sequences of the sequence-coded markers of a data block must be ≤ 256.

6. In the following pseudocode:
    - **dis** is a DataInputStream reading from a bref file.
    - **nSample** denotes the number of samples
    - **nHap** denotes the number of number of haplotypes (**nHap** = 2***nSample**)
    - **nRec** denotes the number of markers with haplotype data in a data block.
    - **nSeq** denotes the number of distinct allele sequences present in the sequence-coded records within a data block
    - **nAllele** denotes the number of alleles (including the REF allele) for a marker.

Pseudocode for reading Bref format:

```
dis = <DataInputStream reading from a  bref file>
snvPerms = <list of lexicographically-sorted permutations of [“A”,“C”,“G”,“T”]>


def readListOfRecords():
  // first read “magic number” and confirm the file format (including version)
  if dis.readInt() != 223579146:
    exit                            // file is not a bref file
  program = is.readUTF()            // program used to create bref file
  nSamples = dis.readInt()          // number of samples
  nHaps = 2*nSamples                // number of haplotypes
  samples = []                      // sample IDs
  for j in range(0, nSamples):
    samples.add(dis.readUTF())

  cumList = []                      // cumulative list of markers with haplotype data
  nRecs = dis.readInt()             // number of records in next data block
  while (nRecs > 0):
    readDataBlock(is, samples, cumList, nRecs)
    nRecs = dis.readInt()
  return cumList


def readDataBlock(is, samples, cumList, nRecs):
  chrom = dis.readUTF()             // CHROM field for all records in data block
  nSeq = dis.readByte() + 128       // number of distinct allele sequences in
                                       sequence-coded records
  hap2Seq = []
  for j in range(0, nHaps):
    hap2Seq.add(dis.readByte() + 128)  // read index of allele sequence carried by
                                          each haplotype at sequence-coded records
  for j in range(0, nRecs):
    rec = readRecord(is, chrom, samples, nSeq, hap2Seq)
    cumList.add(rec)
```

```
def readRecord(is, chrom, samples, nSeq, hap2Seq):

// returns a marker and haplotype allele data in the format:
   (samples, marker, hap2Allele), where
//     samples = is a list of sample identifiers
//     marker = (CHROM, POS, ID, ALLELES, and INFO:END), as in a VCF record
//     hap2Allele = list of numerical haplotype alleles with length nHap2

  marker = readMarker(is, chrom)
  coding = dis.readByte()
  if coding == 0:
    return seqCodedRecord(is, samples, marker, nSeq, hap2Seq)
  else if coding == 1:
    return alleleCodedRecord(is, samples, marker)



def readMarker(is, chrom):
  pos = dis.readInt()            // POS field
  ids = []
  nIds = dis.readByte() + 128    // # of marker IDS
  for j in range(0, nIds):
    ids.add(dis.readUTF())       // read marker IDs
  alleleCode = dis.readByte()    // encodes SNV alleles if alleleCode != -1
  if alleleCode == -1:
    nAllele = dis.readInt()      // number of alleles (including ref allele)
    alleles = []
    for j in range(), nAllele):
      alleles.add(dis.readUTF())
    end = dis.readInt()
    return (chrom, pos, ids, alleles, end)
  else:
    nAllele = 1 + (alCode & 0b11)  // number of alleles (including REF allele)
    permIndex = (nAl >> 2)
    alleles = snvPerms[permIndex][0:nAllele]     // REF is alleles[0]
    end = -1
    return (chrom, pos, ids, alleles, end)
```

```
def alleleCodedRecord(is, samples, marker):
  major = -1
  nAlleles = marker.alleles.length
  hap2Allele = []
  for j in range(0, 2*samples.length):
    hap2Allele.add(-1)
  for j in range(0, nAlleles):
    n = dis.readInt()       // number of haplotypes carrying non-major allele
    if (n != -1):           // allele is non-major allele
      for k in range(0, n):
        hap = dis.readInt()
        hap2Allele[hap] = j
    else:
      major = j
  for j in range(0, 2*samples.length):
    if (hap2Allele[j] == -1):
      hap2Allele[j] = major
  return (samples, marker, hap2Al)


def seqCodedRecord(is, samples, marker, nSeq, hap2Seq):
  seq2Allele = []
  for j in range(0, nSeq):
    seq2Allele.add(dis.readByte() + 128)
  hap2Al = []
  for j in range(0, 2*samples.length):
    hap2Allele.add(seq2Al[hap2Seq[j]])
  return (samples, marker, hap2Alllele)
```