```
import java.io.*;

public class Cache {

  private AVLPuu puu;
  private Jono jono;
  private Log log;

  private long CACHE_SIZE; // max välimuistin koko tavuissa
  private long cacheFreeSpace;
  private int lkm = 0; // tiedostojen lukumäärä välimuistissa

  public Cache(long cacheSize) {
    puu = new AVLPuu();
    jono = new Jono();
    CACHE_SIZE = cacheSize;
    cacheFreeSpace = cacheSize;
  }

  // pyydetään tiedostoa välimuistista
  // logiikka kuvattu LIITE 1:ssä

  public String request(String tiedostonNimi) {

    // tarkastetaan löytyykö välimuistista
    Solmu pyydettyTiedosto = find(tiedostonNimi);

    if (pyydettyTiedosto != null) { // tiedosto välimuistissa
      File tiedostoLevylla = new File(tiedostonNimi);

      if (tiedostoLevylla.lastModified() >
          pyydettyTiedosto.getObject().lastModified()) {

        // levytiedosto uudempi kuin välimuistissa oleva
        // päivitetään välimuistissa olevan tiedoston tiedot
        String palauta = update(pyydettyTiedosto, tiedostoLevylla);
        if (log != null)
          log.write(palauta);
        return palauta;
      }
      else { // tiedostoa ei ole muutettu
        String palauta = update(pyydettyTiedosto); // siirretään jonon perään
        if (log != null)
          log.write(palauta);
        return palauta;
      }
    }
    else { // tiedosto ei välimuistissa
      String palauta = insert(tiedostonNimi); // lisätään välimuistiin
      if (log != null)
        log.write(palauta);
      return palauta;
    }
  }


  // lisätään tiedosto rakenteeseen, jos onnistuu

  private String insert(String tiedostonNimi) {
    File tiedosto = new File(tiedostonNimi);

    if (tiedosto.exists()) { // tiedosto on olemassa
      Tiedosto cacheTiedosto =
          new Tiedosto(tiedosto.getPath(), tiedosto.length(),
                       Lue.tavutTiedostosta(tiedosto),
                       tiedosto.lastModified());
```

```
      if (tiedosto.length() > CACHE_SIZE) // tiedosto liian suuri cacheen
        return "HTTP 1.0/200 OK (too big for cache) " + cacheTiedosto;

      // jos välimuistissa ei tarpeeksi vapaata tilaa
      // poistetaan vanhoja tiedostoja
      if (cacheFreeSpace < tiedosto.length()) {
        while (cacheFreeSpace < tiedosto.length()) {
          // lisätään vapaata tilaa poistetun koolla
          cacheFreeSpace += removeLRU();
        }
      }

      // nyt kaikki ehdot täyttyvät
      // lisätään tiedosto välimuistirakenteeseen
      puu.insert( jono.toQueue(cacheTiedosto) );
      cacheFreeSpace -= tiedosto.length();
      lkm++;
      return "HTTP 1.0/200 OK (fetched to cache) " + cacheTiedosto;
    }
    else { // tiedostoa ei löytynyt levyltäkään
      return "HTTP 1.0/404 Not Found '" + tiedostonNimi + "'";
    }
  }


  // päivitetään tietorakenne onnistuneen haun yhteydessä
  // (siirretään tiedosto jonon perään)

  private String update(Solmu cacheSolmu) {
    Tiedosto cacheTiedosto = cacheSolmu.getObject();
    jono.moveToEnd(cacheSolmu);
    return "HTTP 1.0/200 OK (found in cache) " + cacheTiedosto;
  }


  // päivitetään tietorakenne, jos levyllä uudempi versio
  // (muutetaan tiedot ja siirretään jonon perään)

  private String update(Solmu cacheSolmu, File levyTiedosto) {
    Tiedosto cacheTiedosto = cacheSolmu.getObject();

    // päivitetty tiedosto levyllä suurempi kuin välimuistin koko
    if (levyTiedosto.length() > CACHE_SIZE) {
      // poistetaan vanha tiedosto rakenteesta
      // palautetaan levyltä muutettu tiedosto
      remove(cacheTiedosto);
      Tiedosto palautaTiedosto =
          new Tiedosto(levyTiedosto.getPath(), levyTiedosto.length(),
                       Lue.tavutTiedostosta(levyTiedosto),
                       levyTiedosto.lastModified());

      return "HTTP 1.0/200 OK (modified file too big for cache) " +
             palautaTiedosto;
    }

    // tiedosto jonon perään, jotta se ei joudu poistouhan alle
    jono.moveToEnd(cacheSolmu);

    // päivitellään tietoja
    cacheFreeSpace += cacheTiedosto.getLength();
    cacheTiedosto.setLastModified(levyTiedosto.lastModified());
    cacheTiedosto.setLength(levyTiedosto.length());

    // päivitetty tiedosto suurempi kuin välimuistin vapaa tila
    // poistetaan riittävä määrä vanhoja tiedostoja
    if (cacheFreeSpace < cacheTiedosto.getLength()) {
      while (cacheFreeSpace < cacheTiedosto.getLength()) {
```

```
        // lisätään vapaata tilaa poistetun koolla
        cacheFreeSpace += removeLRU();
      }
    }

    // päivitetään datasisältö vasta kun välimuistissa riittävästi tilaa
    cacheTiedosto.setBytes(Lue.tavutTiedostosta(levyTiedosto));

    cacheFreeSpace -= cacheTiedosto.getLength();
    return "HTTP 1.0/200 OK (updated in cache) " + cacheTiedosto;
  }


  // poistetaan tiedosto välimuistista
  // tarvitaan, jos päivitetty tiedosto liian suuri välimuistiin

  private void remove(Tiedosto cacheTiedosto) {
    jono.remove( puu.remove(new Solmu(cacheTiedosto)) );
    cacheFreeSpace += cacheTiedosto.getLength();
    --lkm;
  }


  // poistaa 'Least Recently Used' -tiedoston välimuistista
  // poistaa jonon keulilta, palauttaa poistetun tiedoston koon

  private long removeLRU() {
    Solmu poistettava = puu.remove( jono.fromQueue() );
    lkm--;
    return poistettava.getObject().getLength();
  }


  // etsii tiedostoa välimuistista

  private Solmu find(String tiedostonNimi) {
    return puu.find(new Solmu(new Tiedosto(tiedostonNimi)));
  }


  // asettaa välimuistin tapahtumien lokitiedoston

  public boolean setLogFile(String lokiTiedosto) {
    try {
      log = new Log(lokiTiedosto);
    } catch (FileNotFoundException e) {
      return false;
    }
    return true;
  }

  // näyttää välimuistin tietoja

  public String getCacheInfo() {
    return "Välimuistin koko: " + CACHE_SIZE + " tavua\n" +
           "Vapaana: " + cacheFreeSpace + " tavua\n" +
           "Tiedostoja välimuistissa: " + lkm + "\n";
  }

  // tulostaa tietorakenteen

  public String toString() {
    return "" + jono + "\n" + puu;
  }

}
```