**Hvordan fungerer det singelton:**

Grundprincippet i Singelton er at der kun kan være en instance af klassen, dvs klassen ikke tillader at flere objekter (instansieringer) eksisterer samtidig.

Alle variationer af Singelton har følgende generelle elementer.

* En **private constructor**, hvilket gør at singleton klassen ikke kan tilgåes gennem instantiering fra andre klasser (kan ikke new’es)
* En **private static attributte** som er en instance af Singleton-klassen
* En **public static metode** der returnere instance af singleton-klassen

**Anvendelse – beskrivelse af steder, hvor det kan bruges – hvilket problem løser pattern?**

Singleton sikrer at en klasse kun har en enkelt instans. Den mest almindelige brug af Singleton er når man skal administrere adgang til delte ressourcer. Det kan fx være adgang til databaser, filer, printere og lignende.

Singleton sikrer, at du kan tilgå instansen overalt i programmet og sikrer instansen mod at blive ændret af anden kode.

(Simpel forklaring: På et system, kan der være flere programmer der udskriver til en printer, men da printeren er langsom, lader man programmerne udskrive til en printer spooler i stedet for direkte til printeren, så programmet ikke skal vente på at printeren bliver færdig.

Printer spooleren administrerer så printkøen. Her er det vigtigt, at alle skriver til den samme spooler, så der ikke opstår flere køer og dermed trafikprop, og derfor kan spooleren være en Singleton.)

**Multi threading**

Ved hjælp af tråde kan flere dele af programmet køre samtidig.

Her under er et code eksempel:

Public class Main extends Thread {

  public static void main(String[] args) {

    Main thread = new Main();

    thread.start();

    System.out.println("Denne code er udenfor tråden");

  }

  Public void run() {

    System.out.println("Denne code er inde i tråden");

  }

}

Tråd programmering kan give et problem. Når flere tråde køre i et Java program så  ønsker vi ikke at trådene laver flere database objekter. Vi vil kun:have et database objekt. Løsningen på det problem er følgende:. .

Java har et Keyword der hedder synchronized. Det kan bruges til at sikre at der kun bliver dannet et database objekt selvom der anvendes flere tråde.

Eksempel:

public static Database getInstance(String n)

{

     if (singleObject == null){

        singleObject = new Database(n);

    }

   return singleObject;

}

public class DatabaseSynchronized

{

   private static DatabaseSynchronized singleObject;

   private int record;

   private String name;

   private DatabaseSynchronized(String n)

   {

       name = n;

       record = 0;

   }.

   public static synchronized DatabaseSynchronized getInstance(String n)

   {

        if (singleObject == null)

      {

           singleObject = new DatabaseSynchronized(n);

       }

   }

   public void editRecord(String operation)

   {

          System.out.println(“Performing a “ + operation +

          “ operation on record “ + record +

         “ in database “ + name);

   }

   public String getName()

      {

            return name;

      }

}

Eksempel:

Hvis flere tråde læser og skriver i den samme variabel så kan der opstå Concurrency problemer. Det vil sige at man ikke kan stole på resultatet.

For at undgå det problem er det en fordel at opbygge koden på en sådan måde, at trådene deler så få attributter som muligt, og sikre at en variabel ikke er i brug i en anden tråd. Til det sidstnævnte formål (at sikre at variablen ikke er i brug i andre tråde) l har Java et keyword der hedder isAlive(). .

Eksempel:

Public class Mail extends Thread {

  public static int amount = 1;

  public static void main(String[] args) {

    Main thread = new Main();

    thread.start();

    // Venter på tråden til den er færdig

    while(thread.isAlive()) {

      System.out.println("Waiting.....");

    }

    // Update amount og skriv dens værdi

    System.out.println("Main: " + amount);

    amount++;

    System.out.println("Main: " + amount);

  }

  public void run() {

    amount++;

  }

}