CSE 403

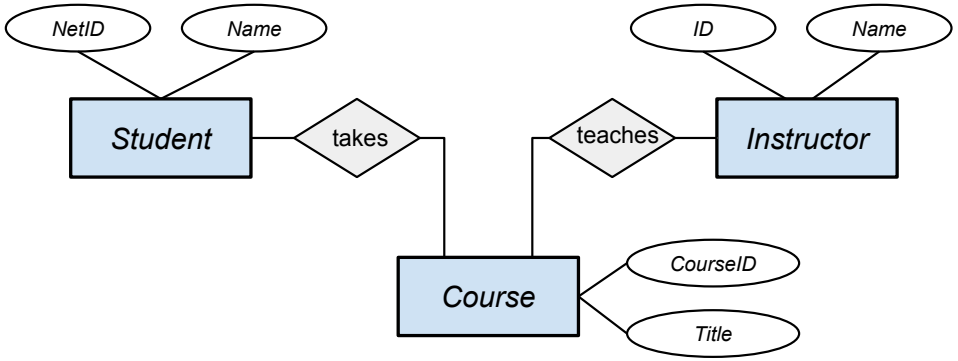
Software Engineering

Software design and best practices

Today

- Recap ER modeling exercise
- A little quiz on best practices

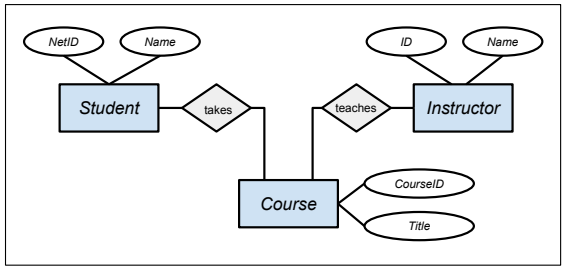
Recap: ER modeling



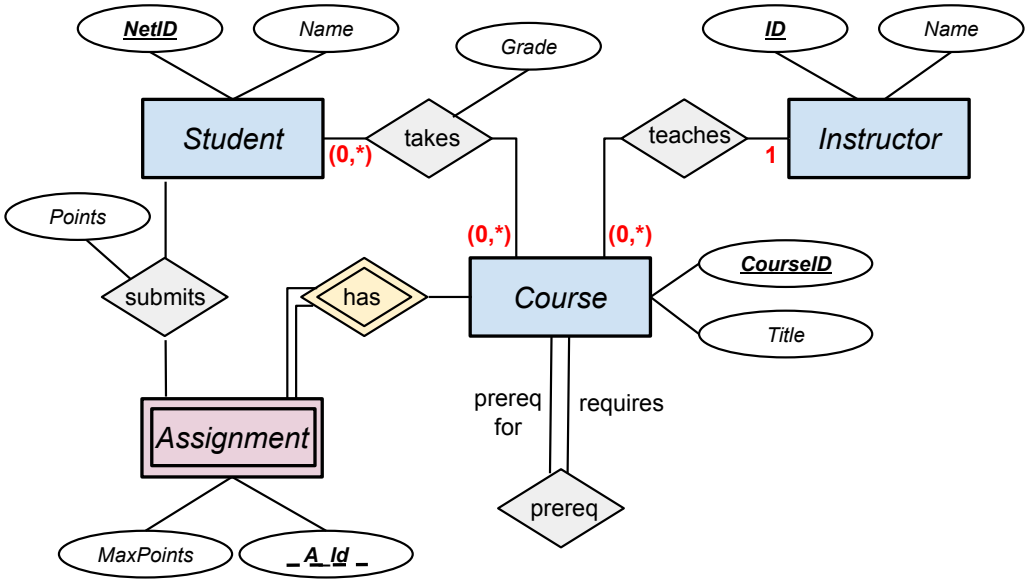
Recap: ER modeling

Let's **augment** our **model** of a course registration system:

- Prerequisites
- Assignments
- Points/grades



Recap: ER modeling



Quiz: setup and goals

- Project groups or small teams
- 6 code snippets
- 2 rounds
  - **First round**
    - For each code snippet, decide whether it represents good or bad practice.
    - **Goal:** discuss and try to reach consensus on good or bad practice.
  - **Second round** (known “solutions”)
    - For each code snippet, try to understand why it is good or bad practice.
    - **Goal:** come up with an explanation or a counter argument.

Round 1: good or bad?

Snippet 1: good or bad?

```
public File[] getAllLogs(Directory dir) {
   if (dir == null || !dir.exists() || dir.isEmpty()) {
      return null;
   } else {
      int numLogs = … // determine number of log files
      File[] allLogs = new File[numLogs];
      for (int i=0; i<numLogs; ++i) {
         allLogs[i] = … // populate the array
      }
      return allLogs;
   }
}
```

Snippet 2: good or bad?

```
public void addStudent(Student student, String course) {
   if (course.equals("CSE403")) {
      cse403Students.add(student);
   }
   allStudents.add(student)
}
```

Snippet 3: good or bad?

```
public enum PaymentType {DEBIT, CREDIT}

public void doTransaction(double amount, PaymentType payType) {
  switch (payType) {
    case DEBIT:
       … // process debit card
       break;
    case CREDIT:
       … // process credit card
       break;
    default:
       throw new IllegalArgumentException("Unexpected payment type");
   }
}
```

Snippet 4: good or bad?

```
public int getAbsMax(int x, int y) {
  if (x<0) {
    x = -x;
  }
  if (y<0) {
    y = -y;
  }
  return Math.max(x, y);
}
```

Snippet 5: good or bad?

```
public class ArrayList<E> {
   public E remove(int index) {
      …
   }
   public boolean remove(Object o) {
      …
   }
   …
}
```

Snippet 6: good or bad?

```
public class Point {
   private final int x;
   private final int y;
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public int getX() {
      return this.x;
   }
   public int getY() {
      return this.y;
   }
}
```

Quiz: setup and goals

- Project groups or small teams
- 6 code snippets
- 2 rounds
  - **First round**
    - For each code snippet, decide whether it represents good or bad practice.
    - **Goal:** discuss and try to reach consensus on good or bad practice.
  - **Second round** (known “solutions”)
    - For each code snippet, try to understand why it is good or bad practice.
    - **Goal:** come up with an explanation or a counter argument.

https://forms.gle/ZNsVktVyar4EwRrg9

Round 2: why is it good or bad?

My take on this

- Snippet 1: bad
- Snippet 2: bad
- Snippet 3: good
- Snippet 4: bad
- Snippet 5: bad
- Snippet 6: good

Snippet 1: this is bad! why?

```
public File[] getAllLogs(Directory dir) {
   if (dir == null || !dir.exists() || dir.isEmpty()) {
      return null;
   } else {
      int numLogs = … // determine number of log files
      File[] allLogs = new File[numLogs];
      for (int i=0; i<numLogs; ++i) {
         allLogs[i] = … // populate the array
      }
      return allLogs;
   }
}
```

Snippet 1: this is bad! why?

```
public File[] getAllLogs(Directory dir) {
   if (dir == null || !dir.exists() || dir.isEmpty()) {
      return null;
   } else {
      int numLogs = … // determine number of log files
      File[] allLogs = new File[numLogs];
      for (int i=0; i<numLogs; ++i) {
         allLogs[i] = … // populate the array
      }
      return allLogs;
   }
}
```

Null references...the billion dollar mistake.

Snippet 1: this is bad! why?

```
public File[] getAllLogs(Directory dir) {
   if (dir == null || !dir.exists() || dir.isEmpty()) {
      return null;
   } else {
      int numLogs = … // determine number of log files
      File[] allLogs = new File[numLogs];
      for (int i=0; i<numLogs; ++i) {
         allLogs[i] = … // populate the array
      }
      return allLogs;
   }
}
```

```
File[] files = getAllLogs();
for (File f : files) {
   …
}
```

Don’t return null; return an empty array instead.

Snippet 1: this is bad! why?

```
public File[] getAllLogs(Directory dir) {
   if (dir == null || !dir.exists() || dir.isEmpty()) {
      return null;
   } else {
      int numLogs = … // determine number of log files
      File[] allLogs = new File[numLogs];
      for (int i=0; i<numLogs; ++i) {
         allLogs[i] = … // populate the array
      }
      return allLogs;
   }
}
```

No diagnostic information.

Snippet 2: short but bad! why?

```
public void addStudent(Student student, String course) {
   if (course.equals("CSE403")) {
      cse403Students.add(student);
   }
   allStudents.add(student)
}
```

Snippet 2: short but bad! why?

```
public void addStudent(Student student, String course) {
   if (course.equals("CSE403")) {
      cse403Students.add(student);
   }
   allStudents.add(student)
}
```

Defensive programming: add an assertion (or write the literal first).
Use constants and enums to avoid literal duplication.

Snippet 3: this is good, but why?

```
public enum PaymentType {DEBIT, CREDIT}

public void doTransaction(double amount, PaymentType payType) {
  switch (payType) {
    case DEBIT:
       … // process debit card
       break;
    case CREDIT:
       … // process credit card
       break;
    default:
       throw new IllegalArgumentException("Unexpected payment type");
   }
}
```

Snippet 3: this is good, but why?

```
public enum PaymentType {DEBIT, CREDIT}
public void doTransaction(double amount, PaymentType payType) {
  switch (payType) {
    case DEBIT:
       … // process debit card
       break;
    case CREDIT:
       … // process credit card
       break;
    default:
      throw new IllegalArgumentException("Unexpected payment type");
   }
}
```

Type safety using an enum; throws an exception for unexpected cases (e.g., future extensions of PaymentType).

Snippet 4: also bad! huh?

```
public int getAbsMax(int x, int y) {
  if (x<0) {
    x = -x;
  }
  if (y<0) {
    y = -y;
  }
  return Math.max(x, y);
}
```

Snippet 4: also bad! huh?

```
public int getAbsMax(int x, int y) {
  if (x<0) {
    x = -x;
  }
  if (y<0) {
    y = -y;
  }
  return Math.max(x, y);
}
```

Method parameters should be final;
use local variables to sanitize inputs.

Snippet 5: Java API, but still bad! why?

```
public class ArrayList<E> {
   public E remove(int index) {
      …
   }
   public boolean remove(Object o) {
      …
   }
   …
}
```

Snippet 5: Java API, but still bad! why?

```
public class ArrayList<E> {
   public E remove(int index) {
      …
   }
   public boolean remove(Object o) {
      …
   }
   …
}
```

```
ArrayList<String> l = new ArrayList<>();
Integer index = Integer.valueOf(1);
l.add("Hello");
l.add("World");
l.remove(index);
```

What does the last call return (`l.remove(index)`)?

Snippet 5: Java API, but still bad! why?

```
public class ArrayList<E> {
   public E remove(int index) {
      …
   }
   public boolean remove(Object o) {
      …
   }
   …
}
```

```
ArrayList<String> l = new ArrayList<>();
Integer index = Integer.valueOf(1);
l.add("Hello");
l.add("World");
l.remove(index);
```

Avoid method overloading, which is statically resolved.
Autoboxing/unboxing adds additional confusion.

Snippet 6: this is good, but why?

```
public class Point {
   private final int x;
   private final int y;

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public int getX() {
      return this.x;
   }
   public int getY() {
      return this.y;
   }
}
```

Snippet 6: this is good, but why?

```
public class Point {
   private final int x;
   private final int y;

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public int getX() {
      return this.x;
   }
   public int getY() {
      return this.y;
   }
}
```

Good encapsulation; immutable object.