1. There are four violations:

1. The `then` keyword is not used in Scala (static error).
2. Local variable `s` must be initialized as part of its definition (static error).
3. The first branch of the `if` evaluates to `Unit`, which does not match `Int`, the type of the second branch and the declared return type of the function (static error).
4. Since `j = -1`, the reference `a(j)` will be out of bounds (checked runtime error).

Note that Scala has no unchecked runtime errors.

2. (a) `int`
   (b) `(a_1 \rightarrow (a_2 \rightarrow a_1))`
   (c) `((\text{bool} \rightarrow a_1) \rightarrow a_1)`
   (d) Not typable: `x` cannot be both a function and an integer.
   (e) `((a_1 \rightarrow \text{bool}) \rightarrow (a_1 \rightarrow \text{bool}))`

3. Since either arm of the `catch` might yield the overall value, the arms must have the same type, but are otherwise unconstrained (just as for an `if` expression). Since `throw` never actually yields a value, it can safely be assigned any type whatever, and we need that flexibility in order to use it in arbitrary positions in the code.

\[ \quad \frac{\pi}{\pi} (\text{throw}) : t \]

\[ TE \vdash e_1 : t \quad TE \vdash e_2 : t \]

\[ \frac{TE \vdash (catch \ e_1 \ e_2) : t}{} \]

4. (a)

OO programmer hacks classes
Functional programmer uses pattern matching

(b)

Scala programmer hacks code
Scala programmer hacks code

5.

```scala
case class P(i:Int, u:T, v:T) extends T {
  def f() = i * u.f() + v.f()
}

case class Q(b:Boolean) extends T {
  def f() = if (b) 1 else 0
}
```
6.a. \( f: x, z \rightarrow g: y. \)

(b)

\[
\text{def } M2(x: \text{Boolean}, y: \text{Int}, z: \text{Int}) =
\begin{align*}
R(w & \Rightarrow \text{if } (x) \ z + w \text{ else } w - 42, \\
& \Rightarrow w + y)
\end{align*}
\]

7. Under method A, \( s \concat “x” \) and \( “x” \concat s \) will both take time proportional to \( |s| \), because the entire string must be copied. Under method B, \( s \concat “x” \) will still take time proportional to \( |s| \), because the string must be traversed, but \( “x” \concat s \) will take only unit time. So comparing the execution times of the following programs should do the trick: if program 1 runs much faster than program 2, method B is being used; if the runtimes are about the same, it’s method A.

Program 1:

\[
s = ””
\]

\[
\text{for } i = 1 \text{ to } 1000000 \text{ do}
\]

\[
s = ”x” \concat s;
\]

Program 2:

\[
s = ””
\]

\[
\text{for } i = 1 \text{ to } 1000000 \text{ do}
\]

\[
s = s \concat ”x”;
\]

8.

\[
\text{def } \text{count}(b: B, x: A) : \text{Int} = b \text{ match } \{
\begin{align*}
& \text{case EmptyB } \Rightarrow 0 \\
& \text{case InsertB}(b, y) \Rightarrow \text{count}(b, x) + (\text{if } (x == y) 1 \text{ else } 0) \\
& \text{case DeleteB}(b, y) \Rightarrow (\text{count}(b, x) - (\text{if } (x == y) 1 \text{ else } 0)) \text{ max } 0 \\
& \text{case UnionB}(b1, b2) \Rightarrow \text{count}(b1, x) + \text{count}(b2, x)
\end{align*}
\}
\]